SLAC – PUB – 4474
November 1987
(A)

INTEGRATED DATABASE APPROACH FOR GEODETIC APPLICATIONS*

ROBERT RULAND
Stanford Linear Accelerator Center
Stanford University, Stanford, California 94309

DETLEV RULAND
Computer Science Dept., University Würzburg, West Germany,
Currently: IBM Almaden Research Center, San Jose, CA 95120

TABLE OF CONTENTS

*Work supported by the Department of Energy, contract DE–AC03–76SF00515.

Invited Talk to be Presented at the International Conference on Statistical and Scientific Database Management, Rome, Italy, June 27–30, 1988

INTEGRATED DATABASE APPROACH FOR GEODETIC APPLICATIONS

ROBERT RULAND

Stanford Linear Accelerator Center
Stanford University, Stanford, California 94309

DETLEV RULAND

Computer Science Dept., University Würzburg, West Germany,
Currently: IBM Almaden Research Center, San Jose, CA 95120

ABSTRACT

Geodetic measurements even of a defined project produce a vast amount of heterogeneous data. The analysis of these data used to be time-and-manpower consuming and only focused on subsets of the data. This paper demonstrates how an integrated database system will provide an immediate standardized and easy access to the entire information support data management, and, consequently, streamline the analysis.

INTRODUCTION

Over the last decade data handling in geodesy and surveying has changed dramatically. What use to be the fieldbook is now a portable computer and the fieldbook keeper has been substituted by an interface. Further down the data processing line one can see the same changes, least-squares adjustments used to require the computational power of mainframes, now, there are a dozen sophisticated program systems available which run on Personal Computers (PC) and provide an even more elegant human interface. Also, there are solutions available for the automated data preprocessing, *i.e.*, for the data handling and preparation from the electronic fieldbook to the creation of input files for the least-squares adjustments /FrPuRRu87/, /RRuFr88/. However, an equally important step has not found much consideration in geodetic discussions and publications, the storage and retrieval of data. This paper will summarize the structure of geodetic data shown at the example of survey engineering data, explains the concepts of relational databases and shows the application of these concepts to survey engineering data.

The data flow between the geodetic data gathering tools and the applications for this data is summarized in Figure 1. First, the readings are stored in measurement instruments, which are prepared by DATA PREPARATION programs yielding a measurement data file for each considered observable. These measurement data are raw measurement data, which must be processed by several PREPROCESSING programs yielding reduced data. Therefore, the preprocessing programs need the calibration data of each instrument. Furthermore, point identifiers are normalized to standard point identifiers, *i.e.*, alias or synonyms are replaced by standard identifiers. The preprocessed measurement data form the input of various DATA ANALYSIS programs, which compute (new) coordinates for the considered points. For each point all sets of new and previously measured coordinates are stored. Each set of coordinates refers to a common or measurement specific underlying coordinate system.

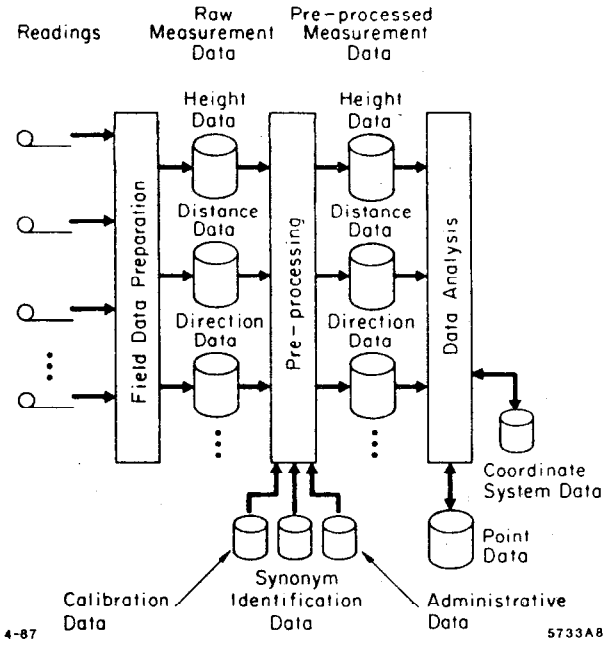


Figure 1

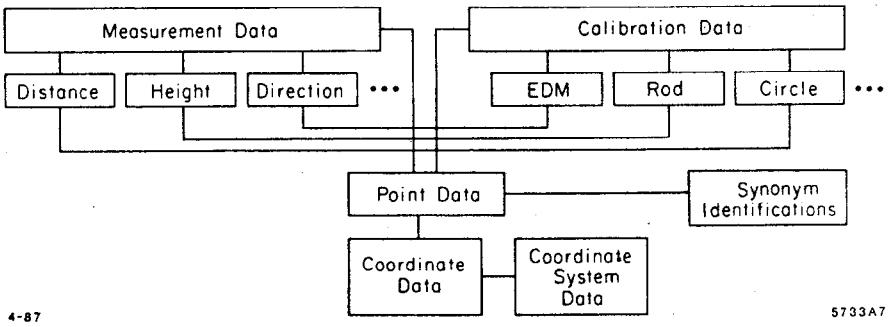


Figure 2

As shown, the geodetic data applications deal with various data which can be classified as follows (see Figure 2):

◇ Measurement Data (Data concerned with different observables):

o Height Data , o Distance Data , o Direction Data .

◇ Calibration Data (Data about instruments):

o Tape Data , o Rod Data , o Circle Data .

◇ Point Data:

o Point Identification Data , o Coordinate Data , o Coordinate System Data .
(Synonym Identifiers)

Summarizing, the described geodetic data applications show a typical file-processing environment. Hence, well-known problems of file-processing environment /Da86/ also appear in geodetic applications: the mentioned geodetic data are shared among various applications, *i.e.*, DATA PREPARATION, PREPROCESSOR and DATA ANALYSIS programs. All these application programs deal with their own data and file structures. As a consequence, replicated data are stored in different structures and file formats at various places, and data consistency problems appear.

Furthermore, the migration of geodetic data applications and tools to further project environments is quite difficult or nearly impossible, because they depend on specific data management characteristics. Thus, geodetic data applications and tools have a limited portability.

INTEGRATED DATABASE APPROACH

As shown above, an integration of the various geodetic data applications and tools is needed. A true integration requires an integrated database support, *i.e.*, a geodetic database, so that data can be shared among applications. All applications access the same geodetic database. Integration provides a unified data management (instead of managing various files), and a centralized data control fully or partially eliminates data redundancy. Moreover, the integrity constraints caused by redundancy of nonremovable data are controlled by the database system. Data sharing means, that different applications can access or manipulate the same part of the centralized database, and conflicting accesses are controlled by the database system. Furthermore, DBMSs provide functions and capabilities, which are also required by geodetic applications (/Da86/, /KoSi87/, /Ul84/).

- ◇ Data Independence

  The applications need not know about data organization, *i.e.*, storage and access structures. Thus, applications are not charged with specific properties and problems of physical data organization and they are free of any hardware details. Furthermore, they are stable against data organization changes.

- ◇ User Views

  The part of the whole database which is of interest to a user is called user view. Selective data base access by the various application is provided.

- ◇ Concurrency Control

  A DBMS controls concurrent access to same data by different applications at the same time.

- ◇ Data Security

  A DBMS provides functions to recover destroyed or lost data due to system crashes or faulty application programs.

- ◇ Access Control and Data Authorization

  A DBMS provides capabilities to hide data to certain users by controlling and granting access rights.

The major advantages of a database approach are:

- ○ Multiuser Access

  Several interactive users and application programs can access the same databases in parallel without knowing about each other. The DBMS controls the concurrent database accesses, that each user/application program feels like accessing the database exclusively.

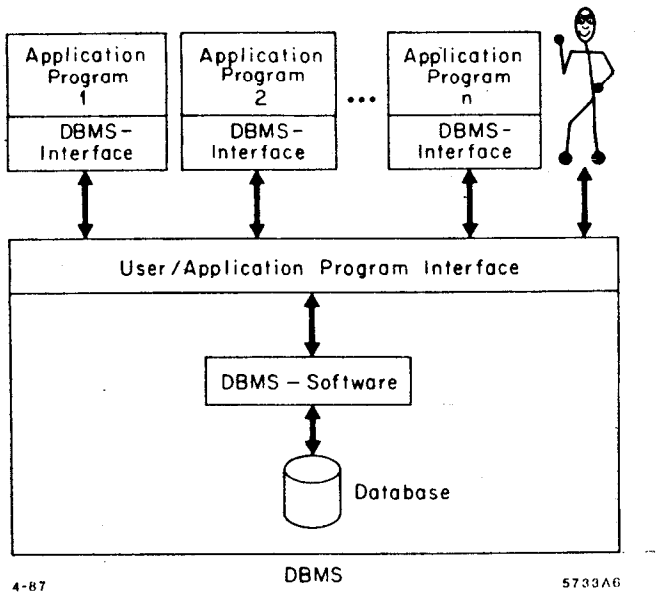


Figure 3

- Uniform DBMS-Interface

  All application programs and tools use the same DBMS–interface. The use of a standardized DBMS–interface increases the portability of tools and application programs, because their DBMS–interface needs not to be changed when migrating among different project environments (Figure 3).

Summarizing, the use of an integrated database approach for the considered geodetic application is sketched in Figure 4. The main advantageous consequences are that the various programs and data gathering tools can be focused on their real geodetic problems, and are independent of any data management problems. This allows a faster and more efficient software development, as well as eases their migration among different project environments.

Database Schemes and Instances

In a database, a collection of information of a considered application (*i.e.*, a part of the real world) is stored and managed. The information of the application has a certain structure, which describes the collection of information (*i.e.*, data collection) of the application, see Figure 5. The structure changes infrequently (or never), whereas the data collection can change by inserting, deleting, or updating certain pieces of data. These modifications must conform to the structure, *i.e.*, the modified data collection must again be compatible with the structure. In a database, the structure is expressed by the DATABASE SCHEME, and the related data collections are called DATABASE INSTANCES. The database instances can only be modified such that they conform to the database scheme.

In an analogy to programming languages, a database scheme corresponds to the type of a variable, and a database instance corresponds to the value of a variable. As the type of a variable describes the feasible values of the variable, the database scheme describes the set of feasible database instances.

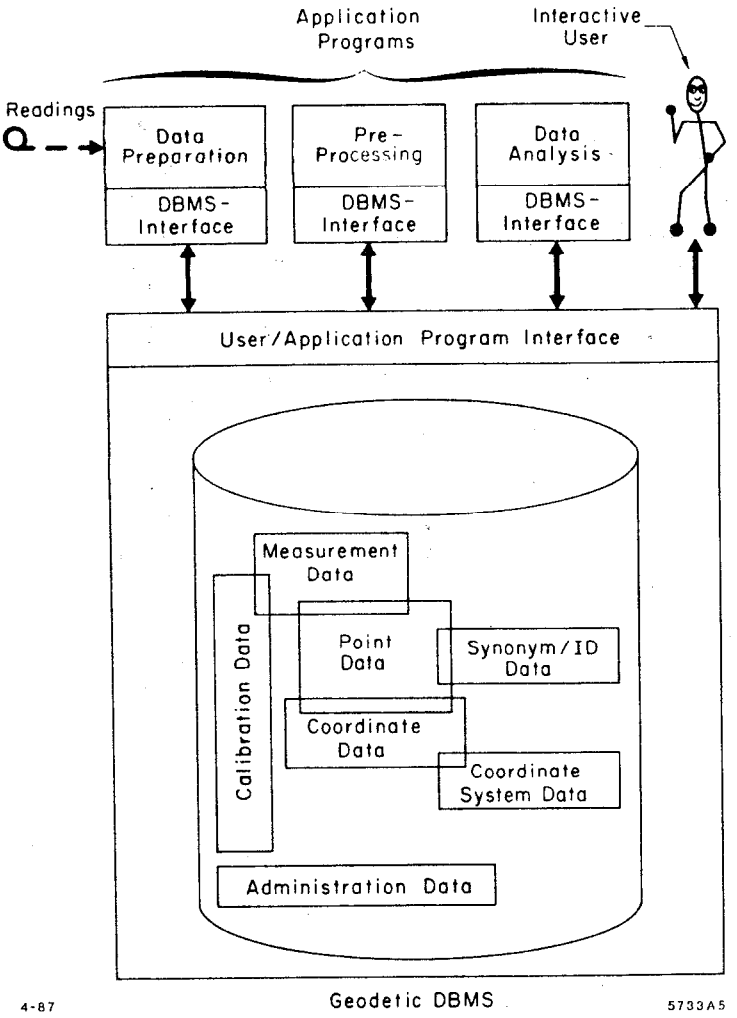


Figure 4

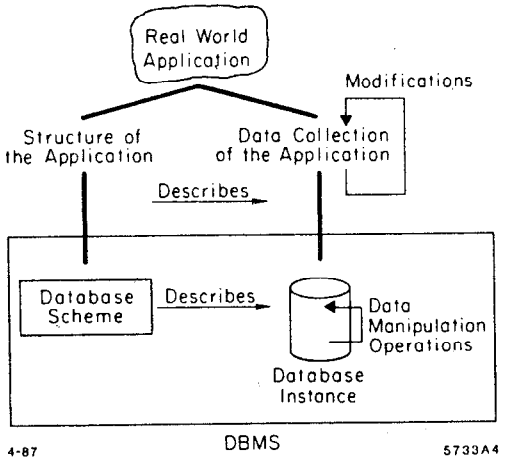


Figure 5

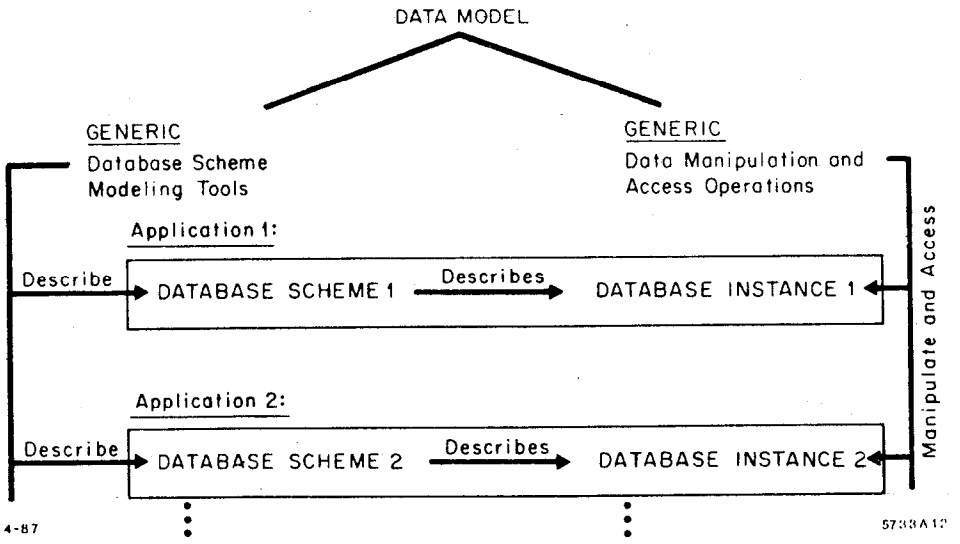


Figure 6

Moreover, a modification of the variable's value yields a new value, which must be compatible with the type of the variable.

Data Models

Each DBMS supports a data model at its user and application program interface. The classical data models are the:

- hierarchical data model,
- network (CODASYL) data model, and
- relational data model.

Commercially available DBMSs are mainly based on these data models. The most frequently used is the relational data model providing a table oriented view of data, which is simple and intuitive.

A data model provides conceptual tools for describing database schemes as well as generic operations to access and manipulate data base instances, see Figure 6.

For example, a relational database scheme is a set of table structures, and a related database instance is the contents of these table structures. Manipulation operations of the relational data model provide insertions, deletions, and updates of rows in tables. Relational access (query) operations generate new tables from the existing tables as described by the query.

The data model of a DBMS is implemented at its user and application program interface by two languages, see Figure 7. The Data Definition Language (DDL) refers to the scheme modelling tools provided by the data model, and the Data Manipulation Language (DML) refers to the operations of the data model. Thus, the DDL is used for defining database schemes, and the DML is used by the database users and application programs for manipulating and retrieving information stored in the database.

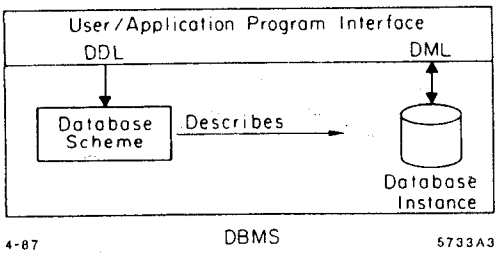


Figure 7

Summarizing, the base of each database application is the design of the (conceptual) database scheme, which reflects the data structures of the considered application, *i.e.*, the objects (entities) and their relationships, which are modelled and managed by the DBMS. Since the database scheme describes the set of possible database instances, the design of the database scheme is of great importance.

As already pointed out, geodetic data are highly structured, *i.e.*, various relationships exist between the geodetic data objects (entities). However, classical data models do not support all types of relationships required by these applications. Those relationship types, which cannot be represented by a data model must be simulated by the supported ones. These limited data modelling capabilities complicate the database design process. This lack of semantics becomes more important the more complex the data structure of the application is (especially in more sophisticated "nonstandard" database applications, like engineering design (CAD/CAM), office automation, geographic applications (land information systems), etc.) /DRuBe87/.

Furthermore, geodetic applications and tools run in a wide range of project environments, which can use DBMSs based on different data models. Thus, the same application data structure must be modelled in the different data models, which causes redundant database design processes.

These gaps between applications and classical data models are bridged by semantic data models, like the Entity/Relationship Model (ER model). The ER model supports all relationship types, and is universal, *i.e.*, generalized from the classical data models. Thus, the data structure of a possible application needs to be modelled only once, and the resulting ER scheme can be algorithmically translated into classical database schemes, see Figure 8.

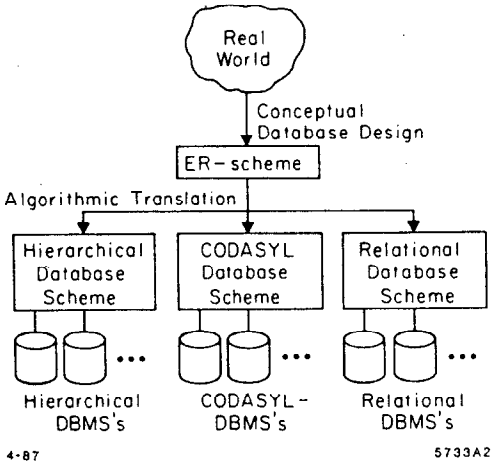


Figure 8

The ER model can be easily extended by more sophisticated data abstraction concepts, like generalizations and aggregation hierarchies, which are often required in nonstandard applications. The use of data abstraction concepts in geodetic applications will be discussed briefly later in this paper.

In the following, the ER model is introduced. The various concepts are explained using the following sample geodetic data classes:

- distance measurement data ,
- height measurement data ,
- calibration data ,
- point data (*i.e.*, coordinates data) .

ER schemes for these data classes are developed and their ER diagrams are given in Figures 10, 11, and 12.

THE ENTITY RELATIONSHIP MODEL

The ER model was introduced by P.C. Chen /Ch76/ in 1976 (see also /Da86/, /KoSi87/, /Ul84/, /Yao85/, to name but a few). Because of its simplicity, the ER model is well-tested and widely accepted. It is based on the concepts of entities and relationships (among entities), which appear to be natural. Since the ER model is a tool supporting the design of a conceptual database scheme, the resulting ER scheme must be translated into a classical database scheme, which can be used as the database scheme for a commercially available DBMS. These scheme transformations can be widely algorithmically performed. The rules for translating ER schemes into relational database schemes will be discussed briefly later.

Entities and Relationships

Conceptually, the ER model describes the given data and its structure using:

▷ entities, ▷ relationships, ▷ and properties of both.

An entity represents "a thing which can be distinctly identified" in the real world. A relationship captures an association among entities. Relevant information about entities and relationships is expressed by their properties.

In the considered geodetic application, height and distance measurements are entities. Properties of a distance measurement are its measurement identifier, the time stamp of the measurement, the reading, and the weather. Properties of a height measurement are also the identifier and the time stamp, as well as the leveled height difference of the two considered points and its probability interval. Moreover, each instrument is represented by an entity. The properties are the instrument identifier and the calibration data. Finally, each point is an entity. The properties of a point are its point identifier, its monumentation, and its hierarchy. Distance measurement entities and point entities are shown in Table 1.

An entity type describes entities with similar properties. Entity types are for instance DISTANCE_MEASUREMENT and HEIGHT_MEASUREMENT, which describe the distance and height measurement entities, respectively. The entity type POINT describes all geodetic points stored in the database. Again, in analogy to programming languages, an entity type refers to the type of a variable, and an entity refers to the value of this variable.

An entity type is specified by its attributes, and the properties of an entity of a certain type are given by values of the entity type's attributes. The attributes of the entity type DISTANCE_MEASUREMENT are DM_Id, T/S, Reading, and Weather. The properties of the sample distance measurement No. 4711 are given by the following attribute values.

TABLE 1

DISTANCE MEASUREMENTS:

4711	14/03/87 @ 9.00	200.00	Rain_Gust
4712	14/03/87 @ 9.05	300.00	Rain_Gust
4713	14/03/87 @ 9.10	400.00	Rain_Gust

POINTS:

P1	Pillar	1
P2	Pillar	2
P3	Plug	1

Attribute	Value
DM_Id	4711
T/S	03/14/87 @ 9.00
Reading	200.00
Weather	Rain_Gust

Entity keys allow the unique identification of entities within an entity set. For instance, the measurement identifiers of distance measurements identify uniquely each distance measurement. Thus, the value of the attribute DM_Id identifies uniquely each entity in any entity set of type DISTANCE_MEASUREMENT. A subset of the attributes of an entity type is called a superkey, if for each entity set, the superkey attribute values of each entity are unique within this set. A candidate key is a superkey with no extraneous attributes, *i.e.*, a minimal superkey. Since there might be several candidate keys, one candidate key must be chosen as the primary key, which is the principal identifier for distinguishing entities within each entity set of this type. The primary key is shortly called key, and attributes, which are not part of the key are called descriptive attributes.

The attribute DM_Id is the key attribute in the entity type DISTANCE_MEASUREMENT, and all other attributes are descriptive attributes.

Entity types with no key attributes are called weak entity types and are considered in the next section. Nonweak entity types are called strong entity types.

Formally, an entity type E is specified by its attribute set V and a subset K of key attributes, *i.e.*, $E = (V,K)$, $K \subset V$.

Furthermore, a domain (value set) $dom(A)$ is defined for each attribute $A \in V$. An entity e of type E is a function, mapping each attribute of the type E onto a value of its domain, *i.e.*,

$$e : V \rightarrow \bigcup_{A \in V} dom(A) \; : \; e(A) \in dom(A); \quad A \in V.$$

Thus, an attribute value (*i.e.*, a property) of an entity must be an element of the domain (value set) of the attribute. The distance measurement listed above is described by an entity e, where

$e\ (DM\_ID) = 4711$, $e\ (T/S) = 03/14/87\ @\ 9.00$,

$e\ (Reading) = 200.00$, $e\ (Weather) = Rain\_Gust$.

An entity set (or instance) $inst(E)$ of an entity type E is a set of entities of type E, such that the key attribute values of each entity in $inst(E)$ are unique in $inst(E)$.

In the following, relationships among entities are considered in more detail. The notions of relationship types and relationship sets are quite similar to those of entity types and entity sets.

TABLE 2

DISTANCE MEASUREMENTS:

4711	14/03/87 @ 9.00	200.00	Rain_Gust
4712	14/03/87 @ 9.05	300.00	Rain_Gust
4713	14/03/87 @ 9.10	400.00	Rain_Gust

POINTS:

P1	Pillar	1
P2	Pillar	2
P3	Plug	1

Starting Point Relationships

There are the following relationships in the considered geodetic application. Each distance and height measurement refers to two points, *i.e.*, the starting and ending point. Thus, each entity of type DISTANCE_MEASUREMENT (and HEIGHT_MEASUREMENT) is related to two entities of type POINT. Relationships associating distance measurements and their starting points are shown in Table 2. Hence, there are relationship types DM_START_POINT and DM_END_POINT among the entity types, DISTANCE_MEASUREMENT and POINT, and HM_START_POINT and HM_END_POINT among the entity types HEIGHT_MEASUREMENT and POINT, respectively. Since a distance can be measured by various methods (tape, EDM, DISTINVAR, Interferometer), each distance measurement entity (of type DISTANCE_MEASUREMENT) must be related to an entity specializing the distance measurement by properties, which are specific for the used method. The common properties of distance measurements, which are independent of any specific method are properties of the distance measurement entity. Thus, there are the entity types TAPE_METHOD, EDM_METHOD, DISTINVAR_METHOD, and INTERFEROMETER_METHOD related to the various methods. Moreover, a relationship type is defined among the entity type DISTANCE_MEASUREMENT and the entity types of the different methods.

As it turns out, in most (geodetic) applications, BINARY relationship types are the most important ones. Binary relationship types involve two (necessarily distinct) entity types. The first entity type is called source entity type, whereas the second one is called target entity type. Several functionalities are defined for binary relationship types, see Figure 9. A functionality can be either

- ◇ one-to-one *(1:1)* ,
- ◇ one-to-many *(1:n)* ,
- ◇ many-to-one *(n:1)* ,
- ◇ many-to-many *(n:m)* .

The functionality of a relationship type defines the following constraints on each relationship set of this type. The *1:1*–functionality expresses, that each source entity can be related to at most one target entity and vice versa. The *1:n*–functionality expresses that each source entity can be related to several target entities but each target entity of the second type can be related to at most one source entity.

The *n:1*–functionality is the dual of the *1:n*–functionality.

The *n:m*–functionality is the weakest one and defines no requirements. Thus, each source entity can be related to several target entities, and vice versa.

For instance, the functionality of the relationship type DM_START_POINT among the entity types DISTANCE_MEASUREMENT and POINT is *n:1*, because each distance measurement has exactly one starting point, but several distance measurements might have the same starting point. The same holds for

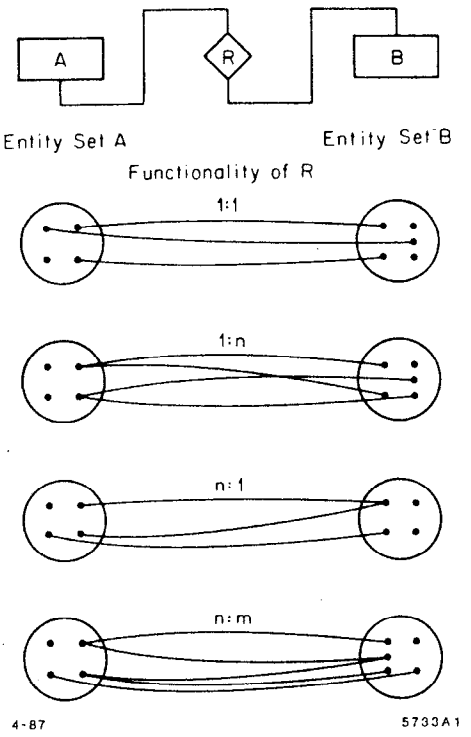


Figure 9

the relationship types among the entity type of each distance measurement method and the entity type of the related instruments. Each measurement can use only one instrument, but several measurements can use the same instrument. There is one exception. A distance measurement based on the DISTINVAR method can use several wires. Thus, the functionality of the relationship type among the entity type DISTINVAR_METHOD and WIRES is *n:m*, because each measurement may use several wires and each wire can be used by several measurements. Finally, the functionality of the relationship type among the entity types POINT and SYNONYM is *1:n*. A SYNONYM entity is an alternate identifier of a point. The functionality is *1:n*, because a point can have several alias identifiers, but each alias identifier is related to exactly one point.

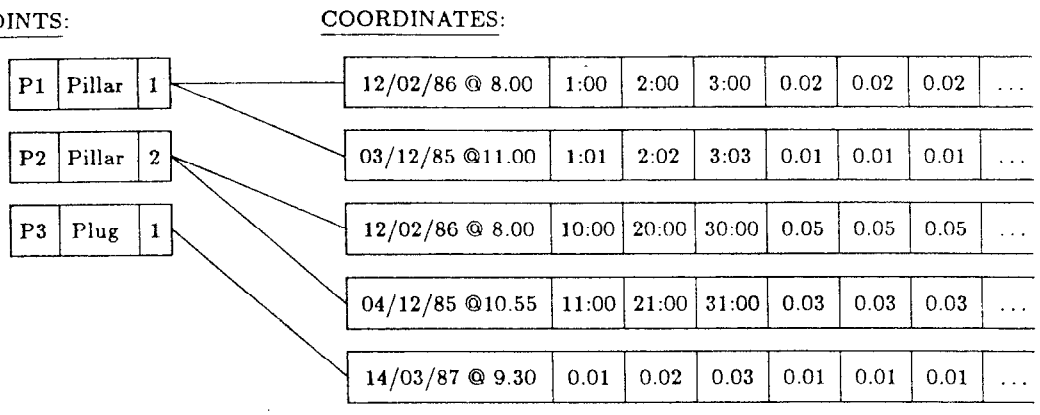
Formally, a binary relationship type R is specified by the two involved entity types E and F, its functionality t, and an attribute set W, *i.e.*, $R = (E,F,t,W)$. The entity type E is called source entity type, and the entity type F is called target entity type. Again, a domain $dom(A)$ must be specified for each attribute $A \in W$. A relationship r of a type R is a function, mapping the source and target entity type onto an entity in their entity sets, and mapping each attribute onto a value of its domain, *i.e.*,

$$r : \{E,F\} \cup W \longrightarrow inst(E) \cup inst(F) \cup \bigcup_{A \in W} dom(A) :$$

$$r(E) \in inst(E), \quad r(F) \in inst(F), \quad r(A) \in dom(A), \quad A \in W.$$

A relationship set or instance $inst(R)$ of a relationship type R is a set of relationships of type R, which additionally satisfies the functionality of R.

TABLE 3. WEAK ENTITY TYPES



ER Schemes and ER Diagrams

Thus, an ER scheme is specified by a set of entity types and a set of relationship types among these entity types. The names of entity types and relationship types must be unique in with an ER scheme. ER schemes are graphically represented by ER diagrams. Entity types are represented by rectangular boxes, and relationship types by diamonds. The connection between an entity type or a relationship type and a related attribute is expressed by a line between the name of the attribute and the box or diamond of the entity type or relationship type, respectively. The key attributes of an entity type are underlined. The connection of a relationship type and the involved entity types is also expressed by a line between the diamond of the relationship type and the box of each entity type. The functionality of a relationship type is indicated at the end of the line at the target entity type box. Sample ER diagrams are given in Figs. 10, 11 and 12. The names of entity types and relationship types are self-descriptive.

Weak Entity Types

Weak entity types are entity types, where no candidate key exists. Especially, weak entity types do not have any primary key.

For instance, COORDINATE is a weak entity type, see Table 3. A coordinate can only exist in conjunction with a point, *i.e.*, there cannot exist any COORDINATE entity, which is not related to a POINT entity.

Thus, weak entities are related to other entities and they are existence dependent on these entities. That means, weak entity types are target entity types of *1:n*–relationship types having the specific semantics of *existence dependency*. Therefore, a weak entity type is called child entity type, and the source entity type of the *1:n*–relationship type is called parent entity type. Moreover, a child entity must be related to *exactly one* parent entity. Thus, no orphans are allowed in any entity set of the child entity type.

Summarizing, a parent entity type is on a higher abstraction level than the related child entity type. Stepping down from a parent entity to its child entities means "collecting more properties" of the parent entity, *i.e.*, the parent entity is refined by the properties of its child entities. The associations among a parent entity and its child entities are called an AGGREGATION HIERARCHY (/SmSm77a/, /SmSm77b/). A relationship type with a weak target entity type is also called PART-OF- or COMPONENT-OF–relationship types.

Nevertheless, the identification problem of weak entities must be considered. Since weak entity types do not have any candidate keys, they cannot be globally identified. However, a unique identification schema is needed to distinguish the child entities of a parent entity in the set of all its child entity, *e.g.*, it is necessary to distinguish the coordinates of a point. This LOCAL identification problem is provided by the concept of local keys or discriminators for weak entity types. Similar to the primary key concept, the discriminator is specified by some attributes of the weak entity type. The discriminator values of a weak entity must be unique within the set of the child entities related to the same parent entity.

For instance, the time stamp of a coordinate identifies locally all coordinates of a point, because there cannot exist two coordinates evaluated at the same time. But there can exist two coordinates of different points, which are evaluated at the same time. Hence, the time stamp of a coordinate is not a global identifier.

Weak entity types are indicated in an ER diagram by doubly outlined boxes. Local key attributes are underlined by a broken line.

Furthermore, the combination of the primary key of the parent entity type and the local key of the child entity type provides a GLOBAL identification of the weak entities. For instance, the time stamp of a coordinate and the identifier of the related point identify this coordinate uniquely in the set of all coordinates.

ER SCHEMES FOR THE SAMPLE GEODETIC APPLICATION

In the following, Figures 10, 11, and 12, ER diagrams are given for distance measurement data, height measurement data, calibration data, and point data. These ER schemes contain 17 entity types and relationship types, respectively. Since entity and relationship types are already introduced and the ER diagrams are self-explaining, only a few aspects are pointed out in the following. The entity types DISTANCE_MEASUREMENT, TAPE_METHOD, EDM_METHOD, DISTINVAR_METHOD, and INTERFEROMETER_METHOD describe the distance measurement data. Notice, that the relationship type DM_METHOD is the only nonbinary relationship type in all considered ER schemes. The entity types HEIGHT_MEASUREMENT and READING describe the height measurement data. The entity types TAPE_INSTRUMENT, EDM_INSTRUMENT, DISTINVAR_INSTRUMENT WIRE, and INTERFEROMETER_INSTRUMENT, as well as ROD_INSTRUMENT describe the calibration data of the instrument used for distance and height measurements, respectively. These entity types are connected to the entity types describing the measurement data. Notice, that the relationship types USED_ROD_1 and USED_ROD_2 are the only relationship types with attributes. The attributes represent the raw and reduced readings on the two scales on each of the two rods used. Finally, the entity types POINT, SYNONYM, COORDINATE, and COORDINATE_SYSTEM describe the point data. Notice, that the relationship type SAME_SERIES is the only recursive relationship type. It relates coordinates, which result from the same measurement epoch.

TRANSLATING ER SCHEMES INTO RELATIONAL DATABASE SCHEMES

The ER model is a tool supporting the database scheme design process. Thus, a developed ER scheme must be translated into an "equivalent" database scheme, which is used for a commercially available

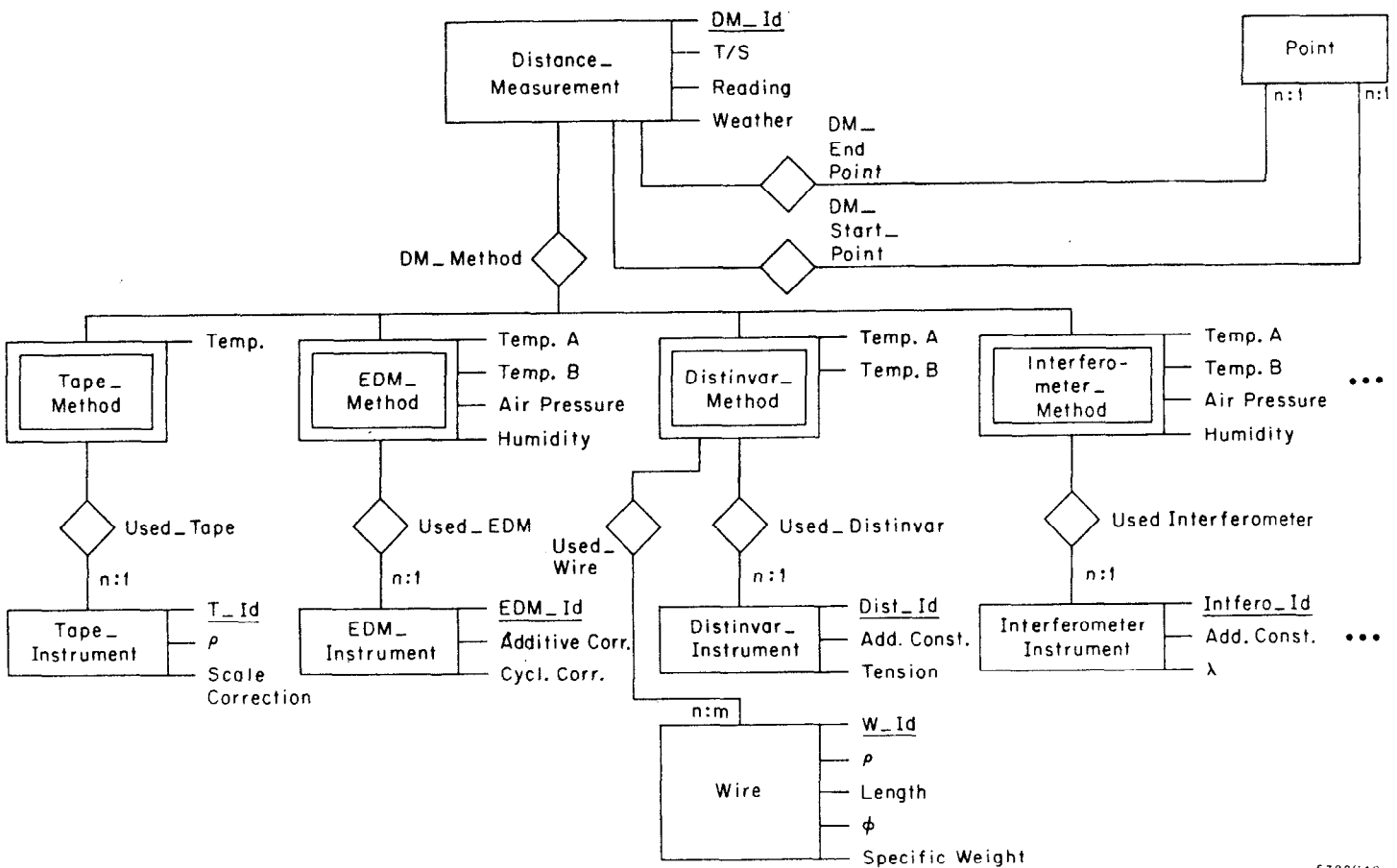

Distance_ Measurement
DM_Id
T/S
Reading
Weather
Point
n:1
n:1
DM_ End Point
DM_ Start_ Point
DM_Method
Tape_ Method
Temp.
EDM_ Method
Temp. A
Temp. B
Air Pressure
Humidity
Distinvar_ Method
Temp. A
Temp. B
Interfero- meter_ Method
Temp. A
Temp. B
Air Pressure
Humidity
Used_Tape
Used_EDM
Used_ Wire
Used_Distinvar
Used Interferometer
n:1
n:1
n:1
n:1
Tape_ Instrument
T_Id
ρ
Scale Correction
EDM_ Instrument
EDM_Id
Additive Corr.
Cycl. Corr.
Distinvar_ Instrument
Dist_Id
Add. Const.
Tension
Interferometer Instrument
Intfero_Id
Add. Const.
λ
n:m
Wire
W_Id
ρ
Length
ϕ
Specific Weight
4-87
5733F10

Figure 10

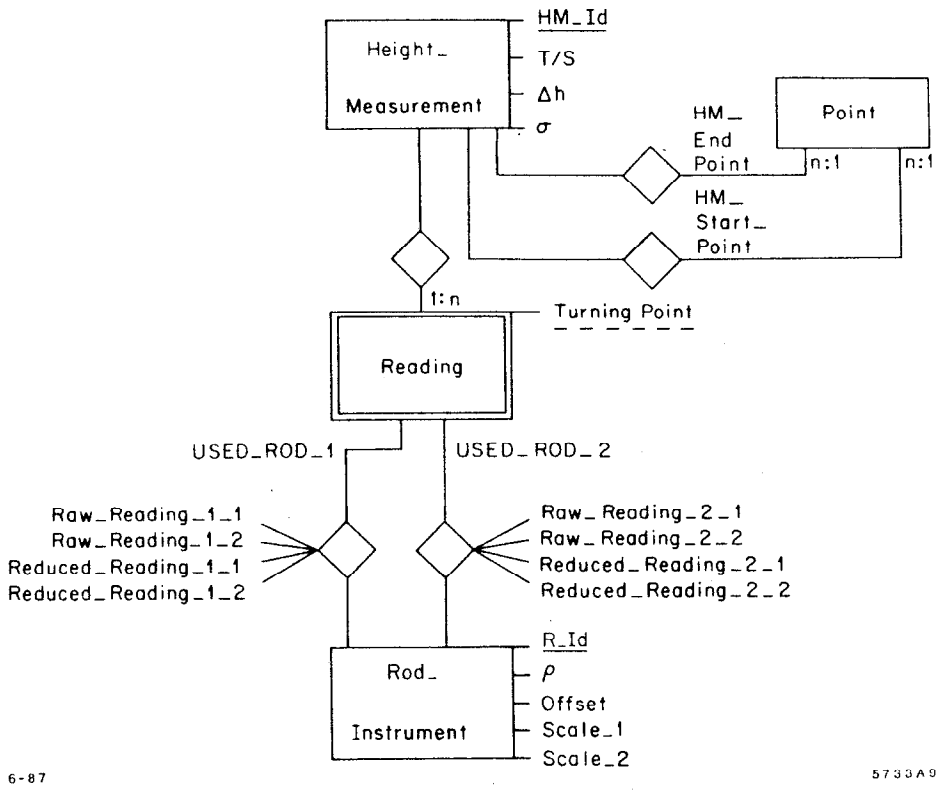


Figure 11

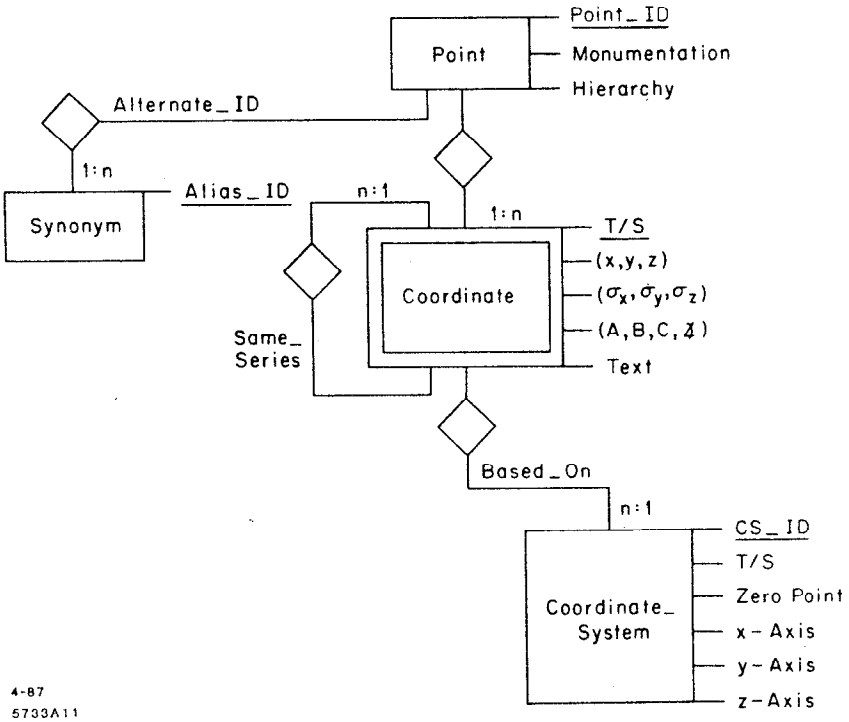


Figure 12

DBMS based on a classical data model. In the following, the translation of ER schemes into relational database schemes is sketched out. As it turns out, these rules can be easily implemented as algorithms. Hence, scheme translations can be automatically performed.

Foundations of the Relational Data Model

The relational data model was introduced by E.F. Codd in 1970 (/Co70/). The basic concept of the relational data model are tables, which correspond directly to the mathematical concept of relations. This allows mathematical investigations and the development of a relational database theory, which is helpful for designing relational database schemes (/Da86/, /KoSi87/, /Mai87/, /Ul84/, to name but a few).

A relational database scheme is a set of relation schemes. A relation scheme RS consists of a set V of attributes, and a subset K of key attributes, *i.e.*, $RS = (V,K)$, $K \subset V$.

Again, a domain is specified for each attribute. A relation is an instance of a relation scheme. A relation of a relation scheme $RS = (V,K)$ is a set of tuples, where each tuple t is a function mapping each attribute A in V onto a value of its domain, *i.e.*,

$$t : V \rightarrow \bigcup dom(A) : t(A) \in dom(A).$$

Furthermore, the key attribute values of t must be unique within the relation.

The "equivalent" relation schemes for the entity types DISTANCE MEASUREMENT and POINT (and corresponding relations) are shown in Table 4.

TABLE 4. RELATION SCHEMES FOR STRONG ENTITY TYPES: COORDINATES

DISTANCE MEASUREMENTS

DM_Id	T/S	Reading	Weather
4711	14/03/87 @ 9.00	200.00	Rain_Gust
4712	14/03/87 @ 9.05	300.00	Rain_Gust
4713	14/03/87 @ 9.10	400.00	Rain_Gust

POINTS

Point_Id	Monumentation	Hierarchy
P1	Pillar	1
P2	Pillar	2
P3	Plug	1

Transformation Rules

Whereas the ER model provides two type constructs (*i.e.*, entity types and relationship types), the relational data model provides only one type construct, *i.e.*, relation schemes. Thus, entity types as well

as relationship types are mapped onto relation schemes, /Ul84/, /YanTeFr85/. First, it is assumed, that all entity types are strong entity types.

An entity type $E = (V,K)$ is mapped onto a relation scheme in a natural way, *i.e.*, the attributes of the entity type define the attributes of the relation scheme, and the key attributes of the entity type define the key attributes of the relation scheme, see Table 4. Since entities and tuples are both attribute functions, an entity is identically mapped onto a tuple.

The relation scheme of a relationship type $RS = (E,F,t,W)$ is defined by the key attributes of the relation schemes of its source and target entity type, and its attributes W, see Table 5. A relationship among two entities is mapped onto a tuple by copying the key attribute values of the source and target entity. Furthermore, the definition of the key in the relation scheme of a relationship type depends on its functionality. If the functionality is *1:1*, then the key attributes in the relation schemes of its source and target entity type define the key. If the functionality is *1:n (n:1)*, then the key attributes in the relation scheme of its target (source) entity type define the key. Finally, no key attributes are defined, if the functionality is *n:m*.

TABLE 5.

RELATION SCHEMES FOR RELATIONSHIP TYPES

DM START POINT

DM_Id	Point_Id
4711	P1
4712	P2
4713	P2

Consider now relation schemes for weak entity types. The relation scheme for a weak entity type are defined by its attributes PLUS the key attributes in the relation scheme of the parent entity type. Thus, a weak entity is mapped onto a tuple similar as a strong entity. Additionally, the key attribute values in the tuple of the parent entity are copied as values of the identical attributes in the child tuple, see Table 6. The key in the relation scheme of a weak entity type is defined by the local key of the weak entity type and the key coming from relation scheme of the parent entity type. Furthermore, NO relation schemes for PART–OF–relationship types are necessary, because the relationship between a child and its parent entity is already expressed in the tuple of the child entity.

TABLE 6. RELATION SCHEMES FOR WEAK ENTITY TYPES

COORDINATES:

T/S	x	y	z	...	Point_Id
12/02/86 @ 8.00	1.00	2.00	3.00	...	P1
03/12/85 @ 11.00	1.01	2.02	3.03	...	P1
12/02/86 @ 8.00	10.00	20.00	30.00	...	P2
04/12/85 @ 10.55	11.00	21.00	31.00	...	P2
14/03/87 @ 9.30	0.01	0.02	0.03	...	P3

FURTHER DATA ABSTRACTION CONCEPTS

Besides aggregation hierarchies, GENERALIZATION is the second data abstraction concept introduced in /SmSm77a/ and /SmSm77b/. Aggregation hierarchies in geodetic applications have already been introduced. Generalization is regarding common properties of entities of different entity types as a new (generalized) entity at a higher level of abstraction. The resulting entity type is called generalized entity type while the original entity types are called individual entity types /DRuBe87/.

The considered geodetic application shows also a generalization, *i.e.*, the relationship type DM_METHOD (Figure 10) is a generalization type. The entity type DISTANCE_MEASUREMENT is the generalized entity type of the individual entity types TAPE_METHOD, EDM_METHOD, DISTINVAR_METHOD, and INTERFEROMETER_METHOD. A DISTANCE MEASUREMENT entity describes the properties of a distance measurement, which are independent of a specific method, *i.e.*, these properties are common to all methods. Whereas the properties of the related individual entity are specific to the used method. Therefore, the attributes of a generalized entity type are called common attributes and are inherited by each individual entity type. Stepping up from an individual entity to its generalized entity means dropping the individual properties, *i.e.*, generalizing from the individual entity. Conversely, stepping down from a generalized entity to its related individual entity means collecting the individual properties and inheriting the common properties, *i.e.*, specialization. Furthermore, for each generalized entity there must exist only one individual entity (of only one individual entity type), and vice versa.

Implementations of generalizations within the relational data model are discussed in /DRuBe 87/.

Finally, an alternative relational implementation of aggregation hierarchies by using SURROGATES for the unique identification of entities and relationships is briefly sketched, *cf.*, /HaLo82/, /LoPl83/, /LoKiMcNPlMei85/. Following this method, a SURROGATE attribute is introduced as the primary key in all relation schemes. The surrogate values are DBMS–wide unique and therefore managed by the DBMS. The users must not modify surrogate values. Using the method described above, the number of key attributes increases with the depth of the entity type in the aggregation hierarchy, because the primary keys of its ancestor entity types are inherited. Following, the surrogate concept, these concatenated keys can be avoided. Only one additional attribute *i.e.*, COMPONENT–OF attribute, is used in the table of a child entity type for referencing the father tuple, *i.e.*, the value of the COMPONENT–OF attribute is the surrogate value of the father tuple. Thus, only one additional attribute is used in the tables of child entity types (independent on the depth of the child entity type in the aggregation hierarchy). The disadvantage of the surrogate concept is, that child entities cannot be addressed and accessed globally, because only their local key attributes are stored in their relation schemes.

CONCLUSION

In this paper, the need of a database approach in geodetic applications is shown. The goal of this paper is:

- to provide some understanding of geodetic activities by an example,
- to evaluate the general potential of database systems (DBMSs) in geodetic applications,
- to introduce data modelling concepts generally used in "nonstandard" applications,

- to show the problems in applying DBMSs in todays' market place for "nonstandard" applications.

Up to now, the geodetic data are stored im different files, which are managed by the file management systems of the various used computer systems. This kind of data management shows various well-known problems, *e.g.*,

- no uniform data access and manipulation interface, *i.e.*, poor enforcement of standards,
- uncontrolled data redundancy yielding data consistency problems,
- no data independence of users and application programs,
- no concurrent multiuser data access, *i.e.*, limited data sharing capabilities,
- no data security, authorization, and recovery mechanisms.

Summarizing, the use of file management systems yields a low programming productivity and excessive program maintenance.

Database management systems (DBMSs) avoid these lacks by supporting a high-level and uniform user interface based on a standardized Data Manipulation Language (*e.g.*, CODASYL–DML or SQL). Furthermore, DBMSs control concurrent multiuser user access in a safe way, as well as provide tools for recovering lost data. Thus, application programs accessing a DBMS are independent of any specific data management details.

The benefits of using DBMSs are:

- minimal and controlled data redundancy,
- data independence of the application programs,
- data sharing among several users or application programs,
- data security, authorization, and recovery mechanisms.

Thus, using DBMSs instead of common file management systems fasts the software development process and eases the migration and maintenance of existing application programs.

However, DBMSs are commonly used in commercial applications, like insurance companies, banks, etc., and not even frequently in mechanical applications. This yields problems in using todays DBMSs in these "nonstandard" applications. One major problem concerns data modelling aspects. Since "standard" applications data are not highly structured, classical data models (hierarchical, network, and relational data model) are sufficient for modelling these database schemes. But, "nonstandard" applications' data are highly structured, *i.e.*, there are a lot of data classes and relationship types between them. As it turns out, the classical data model lacks semantics for "nonstandard" applications. The limited data modelling capabilities complicate the database schema design process. Semantically enhanced data models, like the Entity/Relationship model (ER model), overcome these problems. The ER model is introduced in this paper and its appropriaty is verified by developing sample Entity–Relationship schemes for our sample application.

As mentioned above, todays DBMSs are based on classical data models. Hence, ER schemes must be mapped onto classical database schemes. The derived schemes can be implemented using standard DBMSs. As an example, the mapping of ER schemas onto relational database schemes is discussed in detail. The transformation of ER schemes into classical database schemes can be widely algorithmically performed.

BIBLIOGRAPHY

/Ch76/ Chen, P.P., *The Entity-Relationship Model: Towards a Unified View of Data*, ACM TODS, Vol. 1(1), 1976.

/Co70/ Codd, E.F., *A Relational Model for Large Shared Data Banks*, CACM, Vol. 13(6), 1970.

/Da86/ Date, C.J., *Introduction to Database Systems*, Addison-Wesley, 1986.

/FrPuRRu87/ Friedsam, H.; Pushor, R., Ruland, R., *A Realization of an Automated Data Flow for Data Collecting, Processing, Storing and Receiving*, presented at the ASPRS/ACSM Fall Convention, Reno, Nevada, 1987. (SLAC–PUB–4142).

/Halo82/ Haskin, R.; Lorie, R.A., *On Extending the Functions of a Relational Database System*, Proc. of the Intl. Conf. on Management of Data, ACM, 1982.

/KoSi87/ Korth, H.F.; Silberschatz, A., *Database System Concepts*, McGraw Hill, 1987.

/LOPL83/ Lorie, R.A.; Plouffe, W., *Complex Objects and Their Use in Design Transactions*, Engineering Design Databases, Proceedings of the Annual Meeting, Database Week, San Jose, California, 1983.

/LoKi McNPlMei85/ Lorie, R.A.; Kim, W.; McNabb, D.; Plouffe, W.; Meier, A., *Supporting Complex Objects in a Relational System for Engineering Design Databases*, Query Processing in Database Systems, Springer Verlag, 1985.

/Mai83/ Maier, D., *Theory of Relational Databases*, Computer Science Press, 1983.

/DRuBe87/ Ruland, D., Bever, M.,. *Aggregation and Generalization Hierarchies in Office Automation*, IBM Research Report RJ5660 (57159) San Jose, California, 1987.

/RRuFr86/ Ruland, R.; Friedsam, H., *GEONET–A Realization of an Automated Data Flow*, Invited paper, presented at the XVIII FIG Congress; Toronto, 1986.

/SmSm77a/ Smith, J.M.; Smith, D.C.P., *Database Abstractions: Aggregation and Generalization*, ACM TODS, Vol. 2(2), 1977.

/SmSm77b/ Smith, J.M.; Smith, D.C.P., *Database Abstractions: Aggregation*, CACM, Vol. 20(6), 1977.

/Ul84/ Ullman J.D., *Principles of Database Systems*, Computer Science Press, 1984.

/YanTeFr85/ Yang, D.; Teory, T.; Fry, J., *On the Automatic Transformation of Extended ER-Diagrams into the Relational Model*, Techn. Report, University of Michigan, Computing Research Laboratory, CRL-TR-5-85, 1985.

/Yao85/ Yao, S.B. (ed.), *Principles of Database Design*, Prentice-Hall, 1985.