EMDABS : design and formal specification of a datamodel for a clinical research database system

tu

Research Report
ISSN 0167-9708
Coden: TEUEDE

Eindhoven
University of Technology
Netherlands

Faculty of Electrical Engineering

EMDABS: Design and Formal Specification of a Datamodel for a Clinical Research Database System

by
F.M. Pfaffenhöfer
P.J.M. Cluitmans
H.M. Kuipers

EUT Report 91-E-250
ISBN 90-6144-250-8

January 1991

Eindhoven University of Technology Research Reports
EINDHOVEN UNIVERSITY OF TECHNOLOGY

Faculty of Electrical Engineering
Eindhoven The Netherlands

ISSN 0167- 9708 Coden: TEUEDE

EMDABS : Design and Formal Specification of a Datamodel for a Clinical Research Database System

by

F.M. Pfaffenhöfer
P.J.M. Cluitmans
H.M. Kuipers

EUT Report 91-E-250
ISBN 90-6144-250-8

Eindhoven
January 1991

This report was submitted in partial fulfilment of the requirements for the degree of Master of Electrical Engineering at the Eindhoven University of Technology, The Netherlands.

The work was carried out from May 1990 until December 1990 under responsibility of Professor J.E.W. Beneken, Ph.D., at the Division of Medical Electrical Engineering, Eindhoven University of Technology, under supervision of P.J.M. Cluitmans, Ph.D., and H.M. *Kuipers.*

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

EMDABS: design and formal specification of a datamodel for
a clinical research database system / by F.M. Pfaffenhöfer,
P.J.M. Cluitmans and H.M.Kuipers. - Eindhoven: Eindhoven
University of Technology, Faculty of Electrical Engineering. - Fig.-
(EUT report, ISSN 0167-9708; 91-E-250)
Met lit. opg., reg.
ISBN 90-6144-250-8
NUGI 832
Trefw.: medische informatiesystemen.

SUMMARY

In 1985 the division of Medical Electrical Engineering (M.E.E.) started with the development of a clinical research database system called EMDABS (Electrophysiologic Monitoring DAtabase System) that should store the large amounts of data collected during neuro-electrophysiologic monitoring sessions. This system should make research on this information much easier and should support different clinical monitoring equipment and computer hardware. The database system is developed in the first place for the servo-anesthesia research group of M.E.E. that performs research on automation in anesthesia, but it will be developed in such a way that it is also useful for other neuro-electrophysiologic research.

A datamodel of this database system is needed to specify what information will be stored in the database, what datastructures exist and which constraints (semantics) are forced on the data. A partial datamodel was developed that left the time related information collected during a monitoring session unspecified. This report describes the development of a time related datamodel which is highly flexible and contains all the time related information needed such as equipment settings, information on collected raw data, data derived from raw data and events that took place during a monitoring session. The datamodel is recorded with a formal mathematical technique that is very useful for the recording of these datamodels (especially for the recording of semantics) and a description of this method is given. The formal specification of the total datamodel is presented and also described in this report.

CONTENTS

1. Introduction 1

2. EMDABS : a clinical database system 3

2.1 A short introduction to database systems 3

2.1.1 What is a database system? 3

2.1.2 A relational database system 5

2.1.3 A semantic relational database system 6

2.1.4 The datamodel of a semantic relational database system 7

2.2 An historic review of EMDABS 8

2.3 System requirements of EMDABS 9

3. Semantic relational database concepts and their formal specification in a datamodel 11

3.1 Entities and attributes 11

3.2 Tupletype and tuple constraints 12

3.3 Table, tabletype and table constraints 13

3.4 Database characterisation and database type 14

3.5 Unique identification and keys 15

3.6 Database constraints, subset requirements and foreign keys 16

3.7 Diagrams 18

3.8 Operations on tables 19

3.9 Pascal-like notation of the previous concepts 20

4. Designing a clinical research database 24

4.1 Introduction 24

4.2 Problem analysis 24

4.3 Data analysis 25

4.4 Guidelines to a good datamodel 26

5. The subject-session datamodel 29
5.1 Introduction 29
5.2 Sessions and study's 29
5.3 A short description of the remaining subject-session datamodel 31

6. The time-datamodel 36
6.1 Introduction 36
6.2 The events module 36
6.2.1 Event categories 36
6.2.2 The event list 38
6.2.3 The datamodel of the events module 40
6.3 Equipment and equipment settings 45
6.3.1 The subject-session datastructure of equipment settings 45
6.3.2 A compact structure for equipment settings 47
6.4 The montage and electrodes 50
6.5 The measured physiologic parameters 52
6.5.1 Raw physiologic data 52
6.5.2 Data derived from raw physiologic data 54
6.6 The complete time-datamodel 57
6.7 The complete datamodel of EMDABS 59

7. Conclusions and recommendations 61

Appendix A : Mathematical terms 63

Appendix B : The formal specification of the datamodel 66

References 97

1. INTRODUCTION

The intensity of medical research in our Western Society has grown considerably over the last few decades. Medical knowledge and technological possibilities are still growing and will continue to grow as long as the demands of our society do not change drastically. The use of microtechnology in medical science is one example of the technological improvement that has been made. In the field of neuro-electrophysiologic research a lot has changed as a result of the technological improvements. Many of these developments deal with the automation and monitoring of vital signs [Cluitmans, 1990].

The amount of information that can be recorded and is needed for neuro-electrophysiologic research is enormous. As an example we mention the data that arises from the monitoring of raw EEG signals that can go on for hours. Apart from the information that results from the monitoring of vital signs there is a lot of other information that is needed for neuro-electrophysiologic research. One could for example think of the medical history of subjects that are incorporated in this research, information about the events that occurred during a monitoring session, what sort of equipment was used and how it was adjusted etc.

This large amount of information needed for neuro-electrophysiologic research causes problems for the actual research on this information. To extract the right pieces of information from this large pile of data requires a database system which will make research much easier and surveyable. The accessibility of the information can be highly increased, no piles of papers are needed and the retrieval of information is much faster done by machine than by humans.

Such a database system was required by a multi-institutional neuro-electrophysiologic research group with clinical partners in the Netherlands and the United States [Theisen et al., 1986]. The system should provide for differences in clinical monitoring equipment and computer hardware and should be convenient and easy to use with only minimal computer knowledge. A form of standardized storage of information is necessary to facilitate sharing of data not only among specialists, but also among institutions.

The development of this database system started in 1985 at the division of Medical Electrical Engineering of the Eindhoven Technical University. This database system was called EMDABS (Electrophysiologic Monitoring DAtabase System) and is a relational system. EMDABS is developed in the first place for the servo-anesthesia research group of this division but it should also be useful for other neuro-electrophysiologic research. The servo-anesthesia project deals with the development of automation in anesthesia [Cluitmans, 1990] and one of the objectives of this project is the neurophysiological monitoring of anesthetic depth. Research on anesthetic depth incorporates the analysis of EEG signals in combination with Evoked Potentials (EP), the similarity between different levels of anesthetic depth and sleep patterns and the analysis of the raw EEG signal.

The information collected by this research group during monitoring sessions served as a starting-point in the development of EMDABS. An initial implementation was made in the database management system (DBMS) dBASE III but limitations of dBASE III forced a transfer to the DBMS Oracle [van Herwijnen, 1988]. Limitations in the EMDABS datamodel made a new design of a datamodel necessary. This datamodel specifies which information will be stored in the database, which datastructures exist and what constraints are forced on the data and it is recorded with a formal technique that was developed at the Eindhoven University [de Brock, 1989; Korlaar, 1985; Remmen, 1982].

A partial datamodel was developed which left the time dependent and related information unspecified. This report concerns the design and formal specification of the datamodel and implementation of the table structure in the DBMS Oracle. The time related information includes events recorded with the ERDA (Event Recording and Data Acquisition) system [de Jong, 1986], equipment usage and adjustments, drug administrations, information about raw monitoring data, data extracted from raw monitoring data etc.

2. EMDABS : A CLINICAL DATABASE SYSTEM

In this chapter we will give an historic overview of the development of the clinical database system EMDABS, but first some general concepts of database systems are explained. In chapter 3 a more extensive description is given of semantic relational database concepts and their formal specification.

2.1 A short introduction to database systems

2.1.1 What is a database system?

It is still not possible to give a definition of the term database system that is widely accepted. It seems that every authority on database systems has his own definition of this term and although these definitions are much alike they do tend to lead to confusion. Some writers for example distinguish between the terms 'data' and 'information' while others treat them as synonyms. We can use 'data' to refer to the values physically recorded in the database and 'information' to refer to the meaning of these values as understood by the user. A reason for not distinguishing between the two terms is that they are essentially similar and it is clearer to make the distinction explicit, where relevant, instead of relying on a somewhat arbitrary differentiation between the two. Also the concepts in relation to a database system have differing descriptions. We have chosen some definitions that will give a clear and useful picture of the general concepts.

[Date, 1986] refers to a database system as a computerized recordkeeping system; a system whose overall purpose is to maintain information and to make that information available. This information can be any information that has some significance to the individual or organization the system is intended to serve. In figure 1 we see a simplified picture of a database system which shows four major components: data, hardware, software and users. Database systems can be either multi-user or single-user.

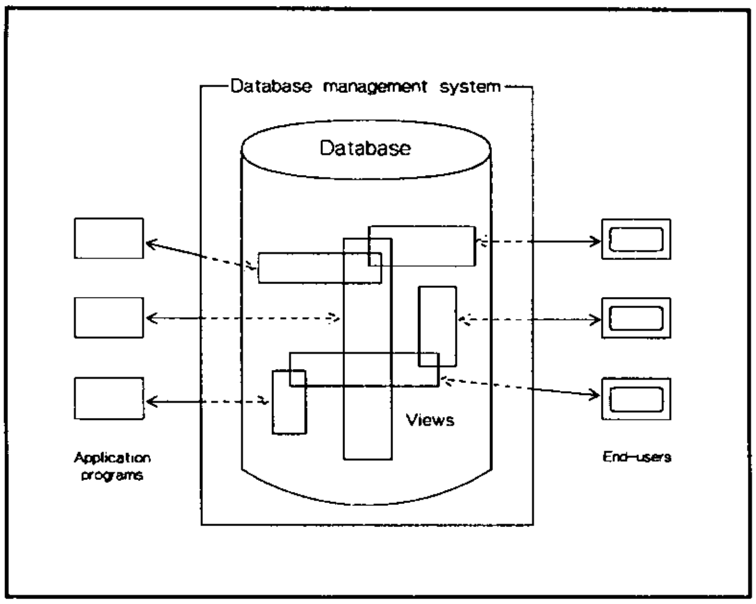

Figure 1 *Simplified picture of a database system*

A database is the centre of a database system and it is a collection of stored operational data used by the application systems of some enterprise which can be a single individual, a large corporation etc. and it is simply used as a convenient generic term. We use the term 'operational data' to distinguish it from input-data, output-data, work queues, temporary results or any purely transient information (information that lasts for a short time only). Input-data may become part of the operational data, but is not initially a part of the database itself. Output-data is data that may be derived from the operational data and this is also not a part of the database itself. The hardware consists of secondary storage volumes on which the database physically resides, together with the associated I/O devices, device controllers etc.

As we can see in figure 1, there is a large layer of software between the physical database and the users of the system. We call this the DBMS (DataBase Management System). The DBMS provides the facilities for creating files (or tables), updating data, inserting data, retrieving data etc. and handles all requests from users for access to the database. The DBMS can thus be seen as a shell that makes the user unaware of the hardware details of the database. It also controls what part of the data a user may have access to, so each user has its own view on the data stored in the database.

Many of the systems provide a built-in user-interface which the end-user can use to access the database. In most cases however the user will access the database via a special application program which is written by an application programmer. An application program can make data-retrieval, data-deletion and data-creation much easier and adjusts it to the user.

Confusion arises when we introduce the term information system. Is a database system not an information system? Yes, but an information system can be much more than that. An information system for example includes full documentation of all procedures and actions which must be taken to get a working application that serves its purpose. Some discord exists about what exactly belongs to the database system and what to the information system but it is clear that there is a distinction between the two [Date, 1986; Hilderink, 1990].

2.1.2 A relational database system

Most database systems developed these days are relational and almost all current database research is based on the relational concept. Other systems such as the hierarchic and the networkmodel exist and we can distinguish the different systems from each other according to the datastructures they present to the user and the operators they provide to manipulate these structures. The hierarchic and network system will not be discussed because they are beyond the scope of this project.

A relational system is a system in which:

1. *the data is perceived by the user as tables (and nothing but tables); and*
2. *the operators at the users disposal are operators that generate new tables from old. For example there will be one operator to extract a column of a given table. This can be regarded as a table itself.* [Date, 1986]

A table consists of information elements which have a certain relation to each other. There will also exist relations between the various tables. An example of a relation between the information elements of a database is given in figure 2. This table represents a list of patients in a hospital and each element contains a piece of information about a patient. A row in a table can be called a record or a tuple (several other terms exist). All elements in a row are thus related to one patient.

PATIENT

Name	address	city	date_of_birth	bloodtype	....
J. Smith	Boschdijk 3	Eindhoven	09-12-1960	O	....
S. Jones	Herenplein 2A	Geldrop	24-04-1951	A	....
....	....	....	....	..	....
....	....	....	....	..	....

Figure 2 *Example of a patient table*

When it is possible to record and guard the various relations on database level then we may speak of a relational database. So besides for creating tables the DBMS must also provide facilities for simultanuously recording relations between tables. If, on the other hand, we need application software to do so, we will have to call it a relational database system. It is much more desirable to have a system that records these relations on database level, where they belong, than to have to use application software. In practice however, most DBMS's provide very little facilities to record relations on database level although the possibilities in new versions are growing rapidly.

2.1.3 A semantic relational database system

In the previous section we have seen what a relational database system is and that it has the possibility to retain various relations between tables or the information elements of a table. Most relational database systems however, 'have only a very limited understanding of what the data in the database means' [Date, 1986]. We would like to be able to incorporate the semantics, the meaning of the data in the database, into the database system. It would then be possible for the database system to recognize that two columns, each extracted from a different table, cannot be joined (see chapter 3.8) because their values are semantically different, although the values both might be numeric values.

In the case of a relational database system where the emphasis lies on the meaning of the data we may speak of a semantic relational database system. In practice this will require the use of application software to record and guard the constraints we have set over the data. Of course we would like to do this on database level instead of having to use application software, just like with all the relations.

2.1.4 The datamodel of a semantic relational database

A datamodel is used to specify which information will be stored in the database, it will show the structures (relations) of the information and the constraints on the information. The datamodel is our basis for an implementation of a database(system). The datamodel is independent of differences in hardware or relational DBMS's.

The constraints on the data are also recorded in the datamodel and we therefore use a formal technique to define our datamodel which is very useful for semantic relational datamodels (see chapter 3).

2.2 An historic review of EMDABS

In 1985 a first beginning was made with the development of a multi-institutional research database for electrophysiologic monitoring of the nervous system. The collaboration of neuro-electrophysiologic research groups from different institutions required a database system that would enable the exchange of high quality information [Theisen et al., 1986]. This database system would also make research much easier on the large amounts of information collected during a monitoring session. This database system was called EMDABS (Electrophysiologic Monitoring DAtabase System) and the development of this database system was primarily done at the division of Medical Electrical Engineering.

A first implementation of EMDABS was based upon dBASE III, an existing DBMS. After some time, limitations of the dBASE III DBMS became clear: in dBASE III a maximum of only fifteen files could simultaneously be opened which was clearly not enough for even the initial datamodel already contained twenty tables [Theisen et al., 1986]. In figure 3 we see the data structure of the first implementation. Because of this and other limitations of dBASE III it was decided not to use this system any more. A larger DBMS was then selected that could handle a large database and could be used on a personal computer and Oracle seemed to be a useful DBMS that could handle the database system in mind (there is also a mainframe version of Oracle). The database implemented in dBASE III was therefore converted to an Oracle version.

With the further development of the database system on Oracle more problems arose, not concerning any limitations of the Oracle DBMS, but which dealt with limitations in the EMDABS datamodel. At that moment it became more and more clear what information was going to be stored in the database and what sort of relations existed between the various kinds of information. It was realized that the existing datamodel was far too simple and that the development of such a datamodel and the database system following from that was highly underestimated. At this point a new start was made with the design of a new complete datamodel.

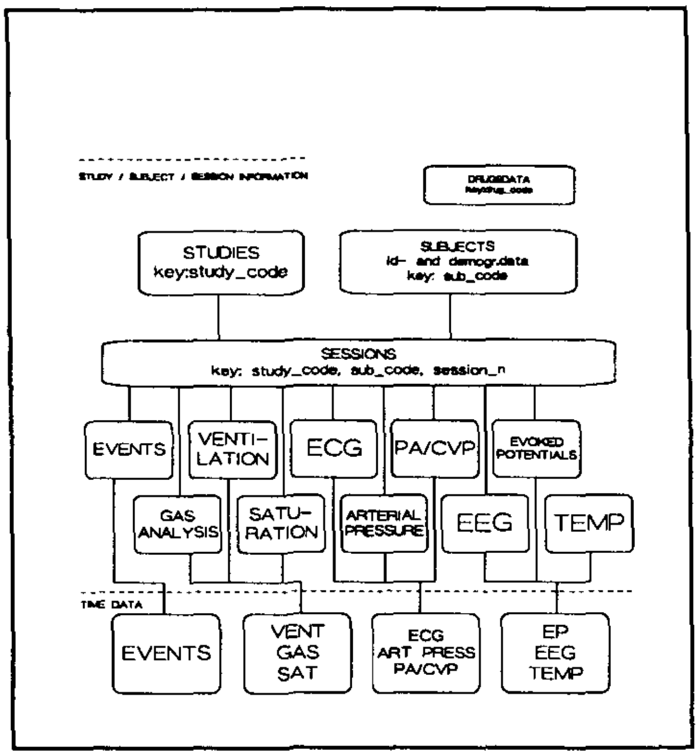

Figure 3 *Datastructure implemented in dBASE III*

In may 1990 when the M. Sc. project described in this report started, a partial datamodel had been designed which left the time-dependent information unspecified. In this project we will develop a datamodel for this time-related information, the time-datamodel, which will be specified by a formal technique.

2.3 System requirements of EMDABS

There have been some changes concerning the system requirements since the beginning of the EMDABS project. The database system was meant to be multi-institutional, which requires a form of standardized storage of information to facilitate sharing data among institutions. The database system that is now being developed will in the first place be used by the servo-anesthesia research group of the division of Medical Electrical Engineering. The information that can be collected in the database however, is more general so that the database system can also be used for other neuro-electrophysiologic research.

The writing of an application program for creating a user-interface that is easy to use will be of great importance. One could for example think of pre-defined questions that can be used to extract information from the database. The main objective at this moment however, is to develop a database system that will make research for the servo-anesthesia group much easier. The database system must therefore have facilities to store large amounts of data including for example the adjustments of equipment, data from arterial bloodpressure, evoked potentials (EP) and the description of real-time events. This may cause hardware problems, but hardware facilities are improving rapidly.

Because of the fact that we want to use the database system for research purposes this will force some heavy constraints on the data in the database. Data integrity has to be controlled and the meaning of the data is of great importance. We want to instruct the database system on the meaning of the data it has stored in its database, so what we are developing is a semantic relational database system. The DBMS Oracle only provides application software to record and guard the constraints on the data and it is not yet possible to do this on database level.

This project is concerned only with the development of the time-datamodel on a database level so we are in fact developing a semantic relational database and we will as such refer to it in the rest of this report.

3. SEMANTIC RELATIONAL DATABASE CONCEPTS AND THEIR FORMAL SPECIFICATION IN A DATAMODEL

In this chapter we will discuss the concepts of semantic relational databases and we will specify them using a formal algebraic technique which will also be used in the formal specification of the datamodel (we will use a pascal-like notation that is equivalent to the more mathematical notation). In appendix A some algebraic terms are defined that will be used in the formal specification. This formal technique is especially useful to record the semantics of the data in our datamodel. This will be done by means of various constraints. By using this formal method we are able to specify the datamodel in a clear and elegant way.

The formal method was developed at the division of Mathematics and Informatics of the Eindhoven University of technology and is described in [de Brock, 1985; de Brock 1989; Korlaar, 1989].

3.1 Entities and attributes

We can distinguish relevant objects within our information system or database system. Possible relevant objects to a hospital are: patient, physician, nurse, treatment etc. Relevant objects to the servo-anesthesia research group are, for example: equipment, monitoring session, study, institution. These relevant objects are called *entities* and each entity is characterized by a set of relevant *attributes*. Relevant attributes to a patient are, for example: name, address, bloodtype etc. And to a monitoring session: session number, begintime, endtime etc.

A regular set function *g* which has the set of entities of an information system as its domain and the sets of attributes of these entities as second coordinates is called the *database skeleton*.

Example:

$$g = \{(\text{article} ; \{\text{code, name, orderaddress}\}),$$
$$(\text{order} ; \{\text{number, date, supplier, amount}\}),$$
$$(\text{supplier} ; \{\text{number, name, address}\})\}$$

For each of the relevant attributes we state what values they may have. We call this an *object characterisation*. An object characterisation of an entity E belonging to a database skeleton *g* is a regular set function over *g*(E). The conditions that the values of separate attributes must meet are called the *attribute constraints.*

For example, let *g*(E)={number, date, supplier, amount}. An object characterisation F of E might be:

$$F = \{(\text{number} ; [1..100000]),$$
$$(\text{date} ; [19000101..20991231]),$$
$$(\text{supplier} ; \text{chs}(15)),$$
$$(\text{amount} ; [1..\ 100000])\}$$

3.2 Tupletype and tuple constraints

We have seen that an object characterisation specifies what values the various attributes can have. We will define a *tupletype* to specify what combination of attribute values are possible for a certain object. A tupletype of an entity E is a non-empty subset of the generalized product of an object characterisation. If we have an object characterisation F and a tupletype T we can formally write this down as:

$$T = \{\ t \mid t \in \Pi(F) \wedge C(t)\ \}$$

where C(t) contains the *tuple constraints* and an element of T is called a *tuple*. The tuple constraints are the conditions each combination of attribute values within a tuple must meet. However, this may not lead to T=∅.If we take the object characterisation F of the previous example we might specify a tupletype T as:

$$T = \{ t \mid t \in \Pi(F) \wedge t(\text{supplier})=\text{'candy factory'} \Rightarrow t(\text{amount})>5000 \}$$

3.3 Table, tabletype and table constraints

We all know what a table looks like (see fig. 2) but we will now give a formal specification of a table. If A is a set then:

$$D \text{ is a table over } A \Leftrightarrow D \text{ is a set of functions over } A.$$

This definition implies that every subset of a tupletype is a table. We now define the term *tabletype* as the set of tables that are interesting to us. Say T is a tupletype then we define a tabletype as:

$$TT \text{ is a tabletype of } T \Leftrightarrow TT = \{ D \mid D \subseteq T \wedge CC(D) \}$$

where CC(D) contains the *table constraints.* These constraints specify which conditions the combinations of tuples within a table must satisfy. Again this may not lead to TT=∅, but TT will contain the empty set (table is empty). If there are no constraints at all then all subsets of the tupletype are useful tables. We give an example that uses the tupletype specified in the previous example:

$$TT = \{ D \mid D \subseteq T \wedge \forall_{t,s \in D} : t(\text{number}) \neq s(\text{number}) \vee t=s \}$$

We will see later in this chapter that the table constraint used here is a special kind of constraint, namely a key. We can also say that TT is a tabletype over A, where A=g(E).

3.4 Database characterisation and database type

In section 3.1 we defined a database skeleton. Now suppose we have the following database skeleton:

$$g = \{(E_1 ; A_1),\ldots, (E_n ; A_n)\} \text{ , } n \geq 1$$

We can now specify the set of possible object occurrences by means of a tupletype and the set of possible tables by means of a tabletype. Analogously we will define a *database type* to designate what combinations of tables actually can occur. Formally we will write this down analogous to the object characterisation and tupletype:

DK is a *database characterisation* to a database skeleton ⇔ Dk is a regular set function over dom(g) and $\forall_{p \in DK}$: k_2p is a tabletype over $g(k_1p)$.

An element of a database characterisation is called a *database snapshot*. We define a database type, also known as a database universe, as a non-empty subset of the generalized product of a database characterisation. More formally:

Let DK be a database characterisation. Then the database type DT belonging to DK is:

$$DT = \{ X \mid X \in \Pi(DK) \wedge DC(X) \}$$

where DC(X) contains the *database constraints*. These constraints are the conditions which the combinations of tables within a database snapshot must satisfy. Again this may not lead to DT=∅.

In the following example the previous formal definitions are used to specify a small database type. This database type is not meant to look like a regular database but is used to give some insight in the previous definitions. We will not use any constraints in order to keep the example simple.

g = {(supplier ; {name, city}), (order ; {supplier, amount})}

F-supp = {(name ; {'Jones', 'Smith', 'Blake'}), (city ; {'London'})}
F-ord = {(supplier ; {'Jones', Smith', 'Blake'}), (amount ; {100, 1000})}

T-supp = Π(F-supp) = {('Jones' ; 'London'), ('Smith' ; 'London'), ('Blake' ; 'London')}
T-ord = Π(F-ord) = {('Jones' ; 100), ('Jones' ; 1000), ('Smith' ; 100), ('Smith' ; 1000), ('Blake' ; 100), ('Blake' ; 1000)}

TT-supp = { D | D ⊆ T-supp } =
{ ∅, {('Jones' ; 'London')}, {('Smith' ; 'London')}, {('Blake' ; 'London')}, {('Jones' ; 'London'), ('Smith' ; 'London')}, ... }
TT-ord = { D | D ⊆ T-ord } =
{ ∅, {('Jones' ; 100)}, {('Jones' ; 1000)}, {('Jones' ; 100), ('Jones' ; 1000)}, {('Jones' ; 100), ('Smith' ; 100)}, ... }

DK-store = {(supplier ; TT-supp), (order ; TT-ord)}

DT-store = Π(DK-store) =
{{(supplier ; {('Jones' ; London)}), (order ; {('Jones' ; 100)})},
{(supplier ; {('Jones' ; London)}), (order ; {('Smith' ; 100)})},
{(supplier ; {('Smith' ; London)}), (order ; {('Blake' ; 1000)})},
{(supplier ; ∅), (order ; ∅)}, ... }

3.5 Unique identification and keys

We do not only want to store the information in a database, we also want to be able to retrieve it. To do this we need some sort of addressing mechanism. We can address a table by its name, but how can we address a specific tuple within that table? To do this we will need an attribute or attributes that will uniquely identify a tuple within a table. This implies that each tuple within a table must have values for these *uniquely identifying* (in short notation: u.i) attributes that differ from the other tuples within the same table. Formally:

if A and B are sets and D is a table over A then:
B is u.i within D $\Leftrightarrow \forall_{t,s \in D} : t \restriction B = s \restriction B \Rightarrow t=s$.

We call B a *key* of a tabletype TT if B is u.i. within each element of TT. If, on top of that, no other real subset of B is a key of TT then we may call B a *minimal key* of TT. More formally:

If A is a set and TT is a tabletype over A then:
B is a key of TT $\Leftrightarrow$ B is a set and $\forall_{D \in TT}$: B is u.i. within D.
B is a minimal key of TT $\Leftrightarrow$ B is a key of TT and $\forall_{C \subseteq D}$:
C is a key of TT $\Rightarrow$ C=B.

The condition that a set B has to be a key of a tabletype TT can be recorded as a table constraint within the tabletype TT.

3.6 Database constraints, subset requirements and foreign keys

Database constraints can connect different tables with each other. The most important class of connect conditions are called the *subset requirements*. There are a lot of different subset requirements so we will take a look at some of their subclasses.

- The standard form of a subset requirement is sometimes called a referential integrity constraint. We can formally specify this constraint as:

If D_1 and D_2 are tables over A and B, and $A' \subseteq A$ and $B' \subseteq B$ are sets then:

$D_1 \Vert\!\restriction A' \subseteq D_2 \Vert\!\restriction B' \infty h$ is a subset requirement. (1)

where h is a bijective function from A′ to B′ that indicates which attribute of A′ corresponds with which attribute in B′. A longer formulation of this subset requirement is:

$$\{ t \upharpoonright A' \mid t \in D_1 \} \subseteq \{ s \circ h \mid s \in D_2 \}$$

When B′ is a key within the D_2 table then we call A′ a *foreign* key. In the case that A and B are equal then h is the identical function and can be omitted. We then get $D_1 \Vert\!\upharpoonright A' \subseteq D_2 \Vert\!\upharpoonright B'$ as our subset requirement.

- A stronger subset requirement arises when the subset requirement works both ways. We then get:

  $D_1 \Vert\!\upharpoonright A' = D_2 \Vert\!\upharpoonright B' \infty h$ is a stronger subset requirement. (2)

The same remarks which apply to the first subset requirement also apply to the second.

- Another sort of subset requirement is an extension of the first two. It specifies, by means of tuple constraints, for which part of the table the requirement must hold. Such a subset requirement could well be:

$$D_1 \Vert\!\upharpoonright A' = \{ t \circ h \mid t \in D_2 \wedge C(t) \}$$

where C(t) contains the tuple constraints.

- The last form of subset requirements we will discuss are actually not database constraints. If the table indices D_1 and D_2 are the same for one of the above subset requirements, then we call this an internal subset requirement. This subset requirement however, can be recorded as a table constraint and therefore cannot be called a database constraint.

3.7 Diagrams

With the recording of subset requirements we formally connect tuples from one table to tuples of another table. We can show these various subset requirements in a diagram to get a compact picture of the datastructure in the datamodel. In figure 4 we see an example of such a diagram. We will now discuss the meaning of the several symbols used in the diagram.

A box in the diagram represents a table and within a box, printed in capital letters, we find the name of the table and the minimal key within a table is written in small print below it. Between the tables there are connection symbols that represent the subset requirements. The connection between table DIVISION and table SPECIALIST for example, represents a subset requirement of form (1). The dotted line tells us that a tuple within the table that is connected to the dotted line, can belong to a tuple within the table on the other side of the connection and a solid line indicates that a tuple within the table connected to the solid line must belong to a tuple within the table on the other side of the connection.

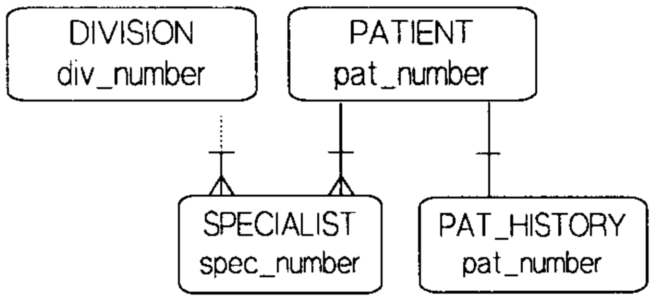

Figure 4 *An example of a diagram*

A tripod means *many* and just one line means *one* to indicate *many-to-one* (1), *one-to-one* (2) etc. relations. If a tripod is used or just one line depends on the set of attributes the subset requirement refers to. If this set is a key within the table we use one line which corresponds with a one-to-... relation and if the set is not a key then we use the tripod, which corresponds with a many-to-... relation.

The relation between the SPECIALIST table and DIVISION table might for example mean that every specialist belongs to a certain division (div_number will be a foreign key within the specialist table), but not every division has to have its own specialist(s). So every div_number in the specialist table must occur in the division table (the division must exist), but not every div_number in the division table may occur in the specialist table.

3.8 Operations on tables

We now know that the information of a relational database system is recorded in tables, so retrieval of this information mostly comes down to the manipulation of tables. Because of the fact that we have defined a table as a set, all known set operations are also applicable to tables. In addition to these operations we define the operation *join* which is used to join two (or more) tables. Formally:

Let D_1 and D_2 be tables then:
$D_1 \bowtie D_2 = \{ t_1 \cup t_2 \mid t_1 \in D_1 \wedge t_2 \in D_2 \wedge t_1$ and t_2 are *joinable* $\}$

What is meant by joinable is defined in appendix A. So if D_1 is a table over A and D_2 a table over B then is $D_1 \bowtie D_2$ a table over $A \cup B$. Some other operations are:

a selection from a table D is specified by:

$\{ t \mid t \in D \wedge C(t) \}$

where C(t) now specifies a selection criterium.

Let D be a table and A a subset of the attribute set of D. The table created by $D \upharpoonright A$ has slunk to a table over A which only contains the attributes we are interested in.

Let D_1 and D_2 be tables over the same attribute set A. Then the union, intersection and difference of these two tables will again be tables.

3.9 Pascal-like notation of the previous concepts

In the formal specification of our datamodel we will use a pascal-like notation that is equivalent to the more mathematical notation of the previous chapters but is more readable and surveyable. We take a small exemplary database extracted from the EMDABS datamodel to show the analogy between the two notations. The bold printed terms of the pascal-like notation have the following meaning:

type : type specification	**endtype** : end of type specification
tatp : tabletype spec.	**endtatp** : end of tabletype spec.
tutp : tupletype spec.	**endtutp** : end of tupletype spec.
obcar : object characterisation	**endobcar** : end of object characterisation
tuc : tuple constraints	**tac** : table constraints
keys : keys of a tabletype	**dac** : database constraints
dbcar : database characterisation	**enddbcar** : end of database characterisation
dbtp : database type spec.	**enddbtp** : end of database spec.

We will first give a formal specification of the exemplary database in pascal-like notation which will be followed by the equivalent mathematical notation. Names of tables, attributes etc. denote the same for both notations.

```
type nr = [1..100 000];
     string = array [1..80] of character;
     body = {'HEAD','NECK','BREAST','ARM','LEG'};
     tatp montage =                          placements of electrodes within a session
       tutp T-mont =
         obcar F-mont =
           session_n      : nr;              session number
           elec_code      : string;          electrode code
           placement      : string;          placement of electrode
           attachment     : string;          means of attachment
         endobcar;
         tuc t(elec_code)='CUP30' ⇒ t(attachment)='GLUE'
       endtutp;
       tac ∀_{t∈D} : ∃_{s∈D} : s(session_n)=t(session_n) ∧ t(attachment)='GLUE'
       keys {{session_n,placement}}
     endtatp;

     tatp electrode_place =                  possible electrode placements
       tutp T-elepl =
         obcar F-elepl =
           placement      : string;          electrode placement
           bodypart       : body;            bodypart on which the electrode is placed
         endobcar;
       endtutp;
       keys {{placement}}
     endtatp;
```

dbtp emdabs =

dbcar DK-em =

mont : montage;

elepl : electrode_place;

enddbcar;

dac $X(\text{mont}) \upharpoonright \{\text{placement}\} \subseteq X(\text{elepl}) \upharpoonright \{\text{placement}\} \wedge$

$\forall_{s \in X(\text{mont})} : \exists_{t \in X(\text{mont})} : \exists_{u \in X(\text{elepl})} : s(\text{session\_n})=t(\text{session\_n}) \wedge$
$t(\text{placement})=u(\text{placement}) \wedge$
$u(\text{bodypart})=\text{'HEAD'}$

enddbtp;

endtype

The equivalent mathematical specification is:

nr = [1..100 000];

body = {'HEAD','NECK','BREAST','ARM','LEG'};

F-mont = {(session_n ; nr) , (elec_code ; Chs(80)) , (placement ; Chs(80)) ,
(attachment ; Chs(80))};

F-elepl = {(placement ; Chs(80)) , (bodypart ; body)};

T-mont = $\{\ t \mid t \in \Pi(\text{F-mont}) \wedge t(\text{elec\_code})=\text{'CUP30'} \Rightarrow t(\text{attachment})=\text{'GLUE'}\}$;

T-elepl = Π(F-elepl)

montage = $\{\ D \mid D \subseteq \text{T-mont} \wedge \forall_{s,t \in D} : ((s(\text{session\_n})=t(\text{session\_n}) \wedge$
$s(\text{placement})=t(\text{placement})) \Rightarrow s=t)$
$\wedge\ \forall_{t \in D} : \exists_{s \in D} : s(\text{session\_n})=t(\text{session\_n}) \wedge$
$t(\text{attachment})=\text{'GLUE'}\ \}$;

electrode_place = $\{\ D \mid D \subseteq \text{T-elepl} \wedge \forall_{s,t \in D} : s(\text{placement})=t(\text{placement}) \Rightarrow s=t\}$;

DK-em = {(mont ; montage),(elepl ; electrode_place)};

$$\text{emdabs} = \{\, X \mid X \in \Pi(\text{DK-em}) \wedge X(\text{mont}) \Vert\!\!\upharpoonright \{\text{placement}\} \subseteq X(\text{elepl}) \Vert\!\!\upharpoonright \{\text{placement}\}$$
$$\wedge\ \forall_{s \in X(\text{mont})} : \exists_{t \in X(\text{mont})} : \exists_{u \in X(\text{elepl})} : s(\text{session\_n}) = t(\text{session\_n}) \wedge$$
$$t(\text{placement}) = u(\text{placement}) \wedge$$
$$u(\text{bodypart}) = \text{'HEAD'} \,\}.$$

4. DESIGNING A DATAMODEL FOR A CLINICAL RESEARCH DATABASE SYSTEM

4.1 Introduction

Designing a datamodel for a database system is a difficult task. The actual successful development of the database system depends on it. The ideal situation would be if a method existed which took us step by step to the actual specification of a good datamodel. For the development of administrative database systems there do exist some methods that guide you through the process of design. Still, a lot will depend on the designer to develop a good datamodel.

We are developing a clinical database system that will be used for research purposes and there is unfortunately not much literature available on the subject of designing clinical research database systems and in particular the design of the datamodel. At the time we started with designing the time-datamodel, there already existed a partial datamodel and a datamodelling experience of about five years. The experience acquired during those years formed a base for a new approach in the development of the time-datamodel. In addition we used relevant guidelines mentioned in [Date, 1986] and our common sense.

4.2 Problem analysis

The first step to be taken in the development of a database system or datamodel is the problem analysis. We can analyze the problem by answering the following questions: what kind of system does a future user need? What are the system requirements? Why is this system needed? What is important to the users of the systems? What are the specific problems of this system?

We will now answer these questions for the EMDABS database system as far as they are related to the design of the datamodel. As we have already stated, EMDABS is a database system that is going to be used for neuro-electrophysiologic research and in particular research related to anesthetic depth [Cluitmans, 1990]. EMDABS will store the large quantities of information that are needed for this research [Theisen et al., 1986]. It will make research on this information much easier, practical and surveyable. An event recording and data acquisition system called ERDA [de Jong, 1986] already existed which collected information during neuro-electrophysiologic monitoring sessions. The datamodel will therefore depend on this recording technique for the data is already structured by this recording system. The meaning of the data in the database is very important for it contains medical information and the information will be used for research purposes. If we want to ensure data integrity, checks must be made on the information as it is entered. We do not want a database filled with contradictory data and nonsense.

As of the varied and unpredictable nature of research needs, our datamodel will be dictated by a high degree of flexibility in integrating data during queries and by the accessibility of data [Budd et al., 1988] [Brower et al., 1984]. Some practical query examples help to get a clear picture of the possibilities that must reside in the data structure. Future extensions and changes of our system should also be considered. One of the requirements stated that the system is intended for research on anesthetic depth but should also be useful for other neuro-electrophysiologic research. Flexibility of our datamodel is therefore important.

4.3 Data analysis

This is the second step we take in the development of the datamodel. Again we can draw up some questions that must be answered: what data will be stored in the database and what is its meaning. What relations exist between the various kinds of data? What are the constraints on the data? What will the data be used for? How is the data collected and what are the consequences?

Data analysis is a difficult process as the data is often spread over a large amount of persons and/or documentation. To be sure of having extracted all relevant information requires a lot of interviews and a good interviewing technique. Asking a lot of (good) questions and frequently feeding back information during design may prevent the design of a datamodel which will be unable to serve its intended purpose. Fortunately, the information we needed was available through one person. We will not discuss the data analysis of EMDABS here, but it is worked up in chapter 6.

4.4 Guidelines to a good datamodel

Within the information we obtained with our data analysis we try to distinguish entities and attributes as to specify a first preliminary database skeleton. An initial datastructure will become clear, but this structure is not nearly definite. Now a lot of what has been said in this chapter becomes important. All knowledge about the system we acquired with the help of our problem and data analysis is needed for designing the datamodel. Apart from that we will need some guidelines to help us with the development of a good datamodel.

Some general guidelines we have used in the design of our model are:

- minimize redundancy. We are dealing with large amounts of information that will need a lot of storage space and redundancy is a waste of this space. More important are the update, delete and insert anomalies that arise from redundancy. If some piece of information exists within more than one table then we will have to be very careful with updating, deletion and insertion of information for this could well lead to a violation of our data integrity requirement. Information that should be exactly the same could now present different values. The introduction of redundancy therefore requires additional checks to insure that data integrity is not violated.

- In some cases however we cannot avoid redundancy because of the fact that it creates the relations between the various tables. An example of this form of redundancy are the foreign keys of the tables. We also accept redundancy if it leads to easier data retrieval, which often means the addition of one redundant attribute to our table. The introduction of redundancy however, must really be compensated by a considerable improvement of data retrieval possibilities.

- Don't let the datastructure be too dependent on the present. For example, if there were only two surgeons present during monitoring sessions in the past, don't make this rigid assumption in your datamodel. It could well be possible that more surgeons will be present in the near future. So again we stress the need for flexibility in our datamodel.

- Keep your datamodel surveyable and neatly structured. If it is impossible to make your way through the maze of tables and relations of your datamodel then there might be something wrong. It will be very hard for possible naive end-users to get insight in the structure of the stored data which will handicap fast and complete use of the database system. It is possible to create too many tables and relations while a simpler compacter structure is possible.

- Try to keep the tables in *Boyce-Codd Normal Form* (BCNF). We will give a definition of BCNF that links up with the formal method described in chapter 3, but first we introduce some other definitions [de Brock, 1989]. We call C *incidental dependent* of B in D if and only if each couple of elements of D that agrees upon B also agrees upon C (for *agree* see appendix A). In a shorter notation we write this down as: B $\rightarrow$ C in D. Do not confuse this arrow with the implication arrow $\Rightarrow$. Formally:

If A,B en C are sets and D is a table over A, then:
$B \rightarrow C \text{ in } D \Leftrightarrow \forall_{t,s \in D} : t \upharpoonright B = s \upharpoonright B \rightarrow t \upharpoonright C = s \upharpoonright C.$

The interesting cases are those where $B \subseteq A$ and $C \subseteq A$ (Formally we don't even need the other cases while $B \rightarrow C$ in $D \Leftrightarrow (B \cap A) \rightarrow (C \cap A)$ in D).

Analogous to incidental dependency we define *structural dependency* for a set of tables. To distinguish this generalized term from the notation $B \rightarrow C$ in D for incidental dependency we use the notation $B \rightrightarrows C$ in TT. Formally:

If A,B and C are sets and TT is a tabletype over A, then:
$B \rightrightarrows C$ in TT $\Leftrightarrow \forall_{D \in TT} : B \rightarrow C$ in D.

The interesting cases are again those for which $B \subseteq A$ and $C \subseteq A$. We can now define the Boyce-Codd normal form:

If A is a set and TT is a tabletype over A, then:
TT is in BCNF $\Leftrightarrow \forall_{B \subseteq A} : \forall_{a \in A} : (B \rightrightarrows \{a\}$ in TT $\wedge\ a \notin B) \Rightarrow$ B is a key of TT.

Another requirement for a tabletype TT to be in BCNF is that it must also be in 1NF (first normal form). 1NF states that every attribute value must be 'atomic'. What we mean here is that an attribute is not allowed to contain a set of values. A problem arises when we use strings as attribute values for a string can be regarded as a set of substrings. However, we will regard a string as being atomic in our datamodel. This assumption has to be made in practice and it won't cause any problems.

Why do we want to keep our tables in BCNF? If an attribute c is determined by a set of attributes B in a tabletype TT ($B \rightrightarrows \{c\} \wedge c \notin B$) and B is not a key of TT then this can lead to update anomalies [Date, 1986]. We can say that c has an alternate key B, but B is not a key within the table of attribute c.

The method we described in this chapter leaves a lot to be done by the designer itself. The designer will need his own common sense to create a datamodel that satisfies the requirements of a good datamodel and of the system.

5. THE SUBJECT-SESSION DATAMODEL

5.1 Introduction

Before we start with the actual design of the time-datamodel we will describe the partial datamodel that already existed. This datamodel includes information about the monitoring sessions, the subjects of the monitoring sessions, personnel present at the sessions etc. and was designed by H. Kuipers of the division of M.E.E. We call this partial datamodel the subject-session datamodel and its diagram is given in figure 5. The diagram technique used here is almost the same as the one described in chapter 3. In figure 5 a connection between two tables can only present a one-to-one relation or a one-to-many relation and nothing more for it was not yet decided, at that moment, what exact diagram was going to be used.

The subject-session datamodel however, has changed during the development of the time-datamodel because of structures that are specified in the time-datamodel. The datamodel also proved to be incomplete and some additional tables had to be created at subject-session level. In this chapter we will only give an extensive explanation of those parts of the subject-session datamodel that are related to the time-datamodel and a give a short description of the rest of the model. A report is written, simultaneously with this report, by H. Kuipers on the design of the subject-session datamodel and will contain a more elaborate description of this model.

5.2 Sessions and studies

A central entity in the total datamodel and thus in the subject-session datamodel is the monitoring session. Our neuro-physiologic research is based upon these monitoring sessions which can involve human subjects as well as animals. A unique session number is assigned to each monitoring session, the attribute session_n, which is therefore the key within the table SESSIONDATA (see fig 5). The table SESSIONDATA contains general information about a monitoring session (subject, type of surgery etc.). Important to the time-datamodel are the

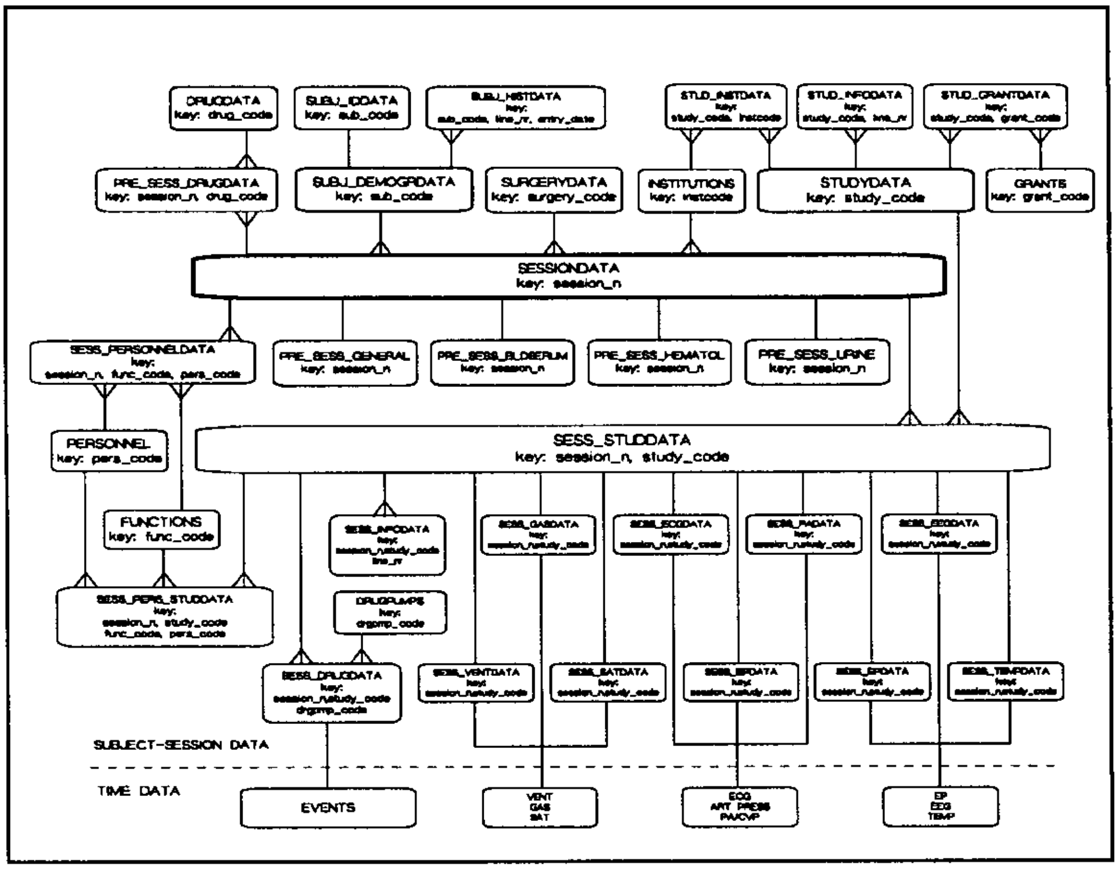
DRUGDATA key: drug_code
SUBJ_IDDATA key: sub_code
SUBJ_HISTDATA key: sub_code, line_nr, entry_date
STUD_INSTDATA key: study_code, instcode
STUD_INFODATA key: study_code, line_nr
STUD_GRANTDATA key: study_code, grant_code
PRE_SESS_DRUGDATA key: session_n, drug_code
SUBJ_DEMOGRDATA key: sub_code
SURGERYDATA key: surgery_code
INSTITUTIONS key: instcode
STUDYDATA key: study_code
GRANTS key: grant_code
SESSIONDATA key: session_n
SESS_PERSONNELDATA key: session_n, func_code, pers_code
PRE_SESS_GENERAL key: session_n
PRE_SESS_BLOSERUM key: session_n
PRE_SESS_HEMATOL key: session_n
PRE_SESS_URINE key: session_n
PERSONNEL key: pers_code
SESS_STUDDATA key: session_n, study_code
FUNCTIONS key: func_code
SESS_INFODATA key: session_n,study_code line_nr
SESS_GASDATA key: session_n,study_code
SESS_ECGDATA key: session_n,study_code
SESS_PADATA key: session_n,study_code
SESS_EEGDATA key: session_n,study_code
SESS_PERS_STUDDATA key: session_n, study_code func_code, pers_code
DRUGPUMPS key: drgpmp_code
SESS_DRUGDATA key: session_n,study_code drgpmp_code
SESS_VENTDATA key: session_n,study_code
SESS_SATDATA key: session_n,study_code
SESS_BPDATA key: session_n,study_code
SESS_EPDATA key: session_n,study_code
SESS_TEMPDATA key: session_n,study_code
SUBJECT-SESSION DATA
TIME DATA
EVENTS
VENT GAS SAT
ECG ART PRESS PA/CVP
EP EEG TEMP

Figure 5 *The subject-session datastructure*

attributes session_n, begin_time and end_time of which the last two attributes represent the begin- and endtime of a session. Time is presented here as a long-integer time that consists of an integer of nine numbers which uniquely identifies a timespot. The long-integer time can be converted to the more readable standard date and time form.

Within a research project that uses the EMDABS system we can distinguish several studies that contain a specific area of neuro-electrophysiologic research. An example could be a study to investigate the effects of certain drugs on neurophysiological parameters performed as part of the anesthetic depth project. The STUDYDATA table in figure 5 contains a short description of each study included in EMDABS. A more extensive description of a each study is found within table STUD_INFODATA. Each study is assigned with a unique (key) attribute study_code.

A monitoring session can be used to acquire data for several studies and a study may include several monitoring sessions. Which session belongs to which study can be found in the SESS-STUDDATA table. Such a table can be regarded as a relation between the SESSION and STUDY table and we will call these tables coupling tables. A tuple within the SESS-STUDDATA table is uniquely identified by the attributes session_n and study_code.

Monitoring sessions can take place at a lot of different institutions. The INSTITUTION table contains information on these institutions (name address etc.) and key within this table is instcode, a unique code assigned to each institution. So table SESSION has a foreign key instcode to denote the institution at which the session took place.

The last table that has a direct relation to the time-datamodel is lookup table DRUGDATA. This table contains a long list of drugs (names, category etc.) which are each uniquely identified by the attribute drug_code. We call this table a lookup table because it contains a long list of certain items that other tables will refer to and thus avoid redundancy. For example, there will be no need to include a full description of a drug each time a table refers to it. A lookup table is also used to force the user to make a choice out of this table when data is inserted, instead of creating his own name for a drug. A lookup table therefore enforces standardisation.

5.3 A short description of the remaining subject-session datamodel

We will now give a short description of the remaining tables in the subject-session datamodel that have no direct relation to the time-datamodel.

SUBJ_IDDATA : This table contains personal information on the subjects of monitoring sessions. The key within this table is sub_code, a unique subject code. The information of this table will have to be protected for it contains confidential information (name, address etc.) on subjects.

SUBJ_HISTDATA : This table contains a general (medical) history of a subject and other information that might be relevant. For example, when a subject is allergic to some drug we can include it in this table. Key within this table is the set of attributes sub_code, line_nr and entry_date. Sub_code represents the subject, line_nr identifies a line of text and entry_date shows the date at which the text line was entered.

SUBJ_DEMOGRDATA : This table contains demographic information on each subject, like: sex, bloodtype etc. Sub_code is the key within this table and as we have stated it represents the subject.

SURGERYDATA : This lookup table contains a list of different types of surgery that have been used in monitoring sessions. Each type of surgery is assigned with a unique code, the surgery_code. This attribute is also a foreign key within the SESSIONDATA table.

STUD_INSTDATA : This coupling table specifies which institution participates in which study. An institution may participate in several studies and a study may be performed by several institutions (a collaboration of institutions is also possible). Key within this table is therefore the set of attributes study_code and instcode. These are keys of the tables being coupled.

GRANTS : This lookup table contains a list of grants that are used for the different studies. Key within this table is the attribute grant_code.

STUD_GRANTDATA : This couple-table specifies which grant belongs to which study. A grant may support several studies and a study may have several grants. Key within this table is the set of attributes study_code and grant_code. Again the key within the couple-table consists of the keys within the tables being coupled, as one would expect.

PRE_SESS_DRUGDATA : This table contains information about drugs that have been administered to the subject before the monitoring session. Key within this table is the set of attributes session_n and drug_code. Drug_code (foreign key) refers to the table DRUGDATA and session_n (foreign key) refers to the table SESSIONDATA.

PRE_SESS_GENERAL : This table contains general information on a subject before a monitoring session like, for example: heart rate, weight and length of the subject etc. Key within this table is session_n which has a one-to-one relation with table SESSIONDATA. The pre-session information belongs to one session only and a session has only one tuple with general pre-session information.

PRE_SESS_BLDSERUM, PRE_SESS_HEMATOL and PRE_SESS_URINE : These three tables contain specific information on a subject which is respectively the labresults on bloodserum, urine and hematologic information of a subject. Session_n is the key within each of these three tables and again this causes a one-to-one relation to exist between these three tables and the SESSION table.

PERSONNEL : This lookup table is in fact a list of all the personnel (with name and address) that have been present at monitoring sessions. Key within this table is the attribute pers_code, a unique personnel code.

FUNCTIONS : This lookup table contains a list of possible functions that personnel attending a monitoring session can have. Func_code is the key within this table.

SESS_PERSONNELDATA : This coupling table specifies what personnel attended each session in what function and it only contains personnel that was needed in general for this monitoring session (as distinct from personnel present for a study). Key within this table is the attribute set session_n, func_code and pers_code which have the same meaning as mentioned above.

SESS_PERS_STUDDATA : This coupling table specifies what personnel attended each session in what function (function is related to the study for which a person is attending a monitoring session) and for what study. Session_n, study_code, func_code and pers_code is the key within this table. It may be possible that one person attends a session on behalf of two different studies in two different function's and is also included in the SESS_PERSONNELDATA table because that person was also needed in general for this session.

SESS_INFODATA : This table contains more extensive information on the coupling of a session and a study in order to keep the SESS_STUDDATA table compact, which will lead to a more convenient use of the SESS_STUDDATA. In most cases we will only want to know which session belongs to which study (see SESS_STUDDATA) and we won't need the extra information. Key within the SESS_INFODATA table is the set session_n, study_code and line_nr. Information is entered as lines of text and each line gets a number, which is why we need the extra attribute line_nr.

The part of the original datamodel we will now explain is merely an indication of how one thought to develop the datastructure of the equipment adjustments and all other time-related data rather than a well-considered datamodel. This part contains some serious flaws that will be dealt with in chapter 6.

DRUGPUMPS : This lookup table contains a list of drugpumps (name, type etc.) that have been used during monitoring sessions. A drugpump is uniquely identified by the attribute drgpump_code.

SESS_DRUGDATA : This table contains information on drugs administered during a session by drugpumps and for which study they were administered. Key within this table is the set of attributes session_n, study_code, drgpump_code. Other drug administrations are found at timedata level (EVENTS).

SESS_GASDATA, SESS_ECGDATA, SESS_PADATA, SESS_EEGDATA, SESS_VENTDATA, SESS_SATDATA, SESS_BPDATA, SESS_EPDATA AND SESS_TEMPDATA : These tables are categorized according to the different physiological parameters that may be measured during a monitoring session and contain the initial settings of the equipment used in relation with these parameters, the equipment names and the study for which these parameters are recorded. Adjustments made during a session are recorded at timedata level. Key within these tables is the set session_n, study_code.

EVENTS : This is not a table but an information block that contains the events recorded with ERDA [de Jong, 1986] during a monitoring session and which will be designed in chapter 6.

VENT, GAS, SAT; ECG, ART PRESS PA/CVP; EP, EEG, TEMP : These three blocks contain the equipment settings adjusted during a session, the collected raw physiologic data and data derived from it. The structure however, is very global and has no real significance as we will see in chapter 6.

6. THE TIME-DATAMODEL

6.1 Introduction

This chapter describes the design of the time-datamodel, but there are also some tables introduced that actually belong at session-subject level but which were needed for the design of the time-datamodel. The diagram technique we will use to show the datastructure of the model is exactly the same as the one described in chapter 3. We will develop the time-datamodel step by step using the subject-session datamodel as a starting point and we will describe this development in an informal way. The formal specification of the time-datamodel can be found in appendix B that contains the formal specification of the total datamodel and which will often be referred to. We will only describe the more complicated constraints of the formal specification which we have marked with a C_N, where n is a number and which we will thus refer to. The other constraints in the specification need no further explanation.

The time-datamodel can roughly be separated in two modules: the events module recorded with ERDA during a monitoring session and the information module related to the automatically recorded physiologic parameters (also recorded by ERDA). We will first start with the design of the events module.

6.2 The events module

6.2.1 Event categories

The datamodel of the events module will depend heavily on the recording system ERDA (Event Recording and Data Acquisition) [de Jong, 1986] that records these events. It is important that we know exactly what took place during an operation (= monitoring session), for each event may influence the physiologic parameters which are automatically recorded and their interpretation. Surgical actions like the incision of tissue, administration of drugs etc. or shifting of the subject will therefore have to be recorded in an event list. The ERDA system has changed since 1986 because of the knowledge acquired by the actual use of the system during monitoring sessions and the events module is designed with respect to these changes.

The ERDA system is implemented on a IBM-compatible personal computer and events can systematically be entered via the keyboard of the computer. One very important aspect of this system is the recording of time that is coupled to these events. There are two time points that are coupled to an event, one automatically registered by the computer and one entered manually. The entry time of an event in the computer is automatically registered in a long-integer time (see chapter 5) and conventional time format (hh:mm:ss). The actual time the event took place has to be entered manually in conventional time format. The recording system couples this time to the event and also adds a long-integer representation of the event time to it so that we now have four time labels coupled to an event. The user-interface of ERDA is menu-driven and allows us to select one of the known events.

The ERDA system distinguishes several major categories in the events that will be recorded on timedata level. These major categories are:

Drugs :	Two subcategories can be distinguished in drugs: drug concentrations and drug administrations. During a monitoring session drugs can be administered in three ways: continuously (drugpumps), intermittently (injections, tablets) and in gas form. The concentration of the administered drugs are also recorded and may change during a session. It is also possible that the same drug is administered in different ways.
Fluids :	The fluids that leave or enter the body are also recorded during a monitoring session. The loss of blood, urine etc. is for example recorded and the administration of glucose solution, ringer's solution, saline etc. Samples may be taken from the fluids which leave the body to be analyzed in a laboratory. The results from these analyses

will also be recorded on timedata level.

Events : These are the operative actions that took place during a monitoring session or events related to anesthesia. Some of these events are: start of anesthesia, intubation, start operation, incision, closing wound, movement, estimation of anesthetic depth etc. Later on in this chapter we will see that the estimation of anesthetic depth does not really belong in this category.

Commentary : We can use commentary to describe events that can't be categorized according to the standard events known by the system. It is therefore possible to enter text line by line and we use this for example, to record commentary made by the anesthesiologist.

Artifacts : Something might happen during an operation which is known to cause disturbances in one or more of the recorded physiologic parameters and we call these events artifacts. A correct interpretation of the measured parameters is only possible if we record these events so there will be no need to guess for what might have caused the disturbance. Some known artifacts are: electrosurgery, movements, interference from power supplies, eye blinking.

6.2.2 The event list

The output of the event recording system is written to a background memory and this output is used to generate an event list so that we are able to read over everything that has happened during a monitoring session. Each event is provided with a unique number that makes the searching of these lists more convenient.

The recording format of the event list exists of a list of output lines, each line representing an event and containing several different fields of which the first four fields are always used for the recording of time. The first field represents the long-integer time at which the event was entered, the second is the matching normal time form in hh:mm:ss. The third field represents the long-integer time at which the event actually took place and the fourth field is again the matching normal time form in hh:mm:ss.

The fifth field contains a code that specifies an event type or category; DC might for example mean: continuous drug administration. The sixth field contains the unique event number that allows us to exactly define the event that took place as well as the event type. In the seventh field we find a description of the event that was menu-selected and it usually contains a copy of the menu-line. In the case of a commentary event this field will contain the entered text of the comment.

After the seventh field may follow another number of fields, but this depends upon the event specified in this output line. We will now describe what additional fields belong to what event type:

- Drug administration continuously and intermittently: drug number, unit number, administered dose, cumulative dose.
- Drug administration gasses : drug number, unit number , administered dose.
- Drug concentrations : drug number, unit number, concentration.
- Fluids : unit number, administered/loosed dose, cumulative dose.
- Known artifacts and events : no additional fields.
- Estimation of anesthetic depth : estimation value.

Drug number and unit number are codes that represent a drug and a unit respectively. We also mention that events which have been entered incorrectly are also recorded in the event list while a correction is followed later. The incorrectly entered events have to be stored in our database as well because of strict medical regulations. An example of a part of an event list is given in figure 6.

```
EVENT LIST nr: 3; DATE & TIME: Thu Jul 06 12:01:28 1989

==============================================================================
  ltime   |  time    |  ltime   |  time    |code| nr | event
==============================================================================
@
615754899; 12:01:39; 615754899; 12:01:39;23;NLA not connected or inactive
615754900; 12:01:40; 615754900; 12:01:40;23;current trigger type: NOG NIET GEINITIALISEERD
615754968; 12:02:48; 61575468  1;  91; ALFENTANIL          ; 79; 1; 3500.00; 3500.00;
***CORRECTION****
615755035; 12:03:55; 615754968; 12:02:48;   1;  91; ALFENTANIL          ; 79; 1; 3.50; 3.50;
615755103; 12:05:03; 615755103; 12:05:03;   4;  16; INCISIE                    ;
615755110; 12:05:10; 615755110; 12:05:10;  13;  21; SCHATTING ANESTH.DIEPTE; 4;
615755121; 12:05:21; 615755121; 12:05:21;  23; trigger off
615755139; 12:05:39; 615755139; 12:05:39;   7; 117; meting gestopt vanwege diathermie  ;
615755343; 12:09:03; 615754200; 11:50:00;   1;  92; VECURONIUM          ; 125; 1; 7.00; 7.00;
615755385; 12:09:45; 615754200; 11:50:00;   1;  99; PROPOFOL            ; 0; 1; 70.00; 70.00;
615755411; 12:10:11; 615755411; 12:10:11;  23;trigger on:d:\eegdata\R101pet.4
615755406; 12:10:06; 615754200; 11:50:00;   0;  78; ALFENTANIL          ; 79; 4; 3750.00; 0;
615755418; 12:10:18; 615755418; 12:10:18;  23;trigger off
615755419; 12:10:19; 615754200; 11:50:00;   0;  79; VECURONIUM          ; 125; 4; 3000.00; 0;
615755430; 12:10:30; 615754200; 11:50:00;   0;  80; PROPOFOL            ; 0; 10; 370.00; 0;
615755491; 12:11:31; 615755491; 12:11:31;  23;please enter depth of anesthesia
615755545; 12:12:25; 615755545; 12:12:25;  13;  21; SCHATTING ANESTH.DIEPTE; 4;
```

Figure 6 *An example of a part of an event list*

6.2.3 The datamodel of the events module

The information found in an event list has to be stored in our database. The problems we have to cope with are the following:

- The user has to be able to extract a complete survey of events that occurred during an operation without much effort.
- The event list contains fields that have different meanings for different event types.
- Incorrect events must be stored as well.
- The values of doses, anesthetic depth and concentrations will be used in calculation processes.

Because of the fact that the user has to be able to extract a complete survey of the events from our database we create a table EVENTS (formal specification: see appendix B). This table however, will not contain the additional fields that have more than one meaning, for the number of additional fields vary for each event type and it would be difficult to force constraints on these fields because of a difference in semantics. The fact that these fields may be used for calculations also requires that we must be sure that the fields we use have the same meaning. The diagram of the events module is given in figure 7.

We can only include the first seven fields of the events list in our EVENTS table. First we remark that monitoring sessions can take place at the same time which will lead to event lists containing equal long-integer times. We can uniquely identify an event if we know the long-integer entry time of an event (it is impossible to enter two events in the computer at the same time) and the session number, so we include these attributes as the key within our table. We can't use the long-integer time at which the event took place to uniquely identify a tuple, because events can take place at the same time (not unique) and we can't use the conventional time form of the entry time because it specifies only time and no date.

The first five attributes of the EVENTS table are: *session_n* and the four time fields *long_intime*, *intime*, *long_evtime*, *evtime* in the same order as found in the event list. *session_n* is a foreign key to the SESSIONDATA table and there is always an event list of a session (C11). The other attributes represent the event number (*ev_number*), event description (*event*), event type description (*eventtype*) and correct entry or not (*correct*). The event number is included because numbers are more convenient for searching databases than text strings. The event type represents the different event categories we get when we distinguish events according to the additional fields as seen in chapter 6.2.2. Two relations between the EVENTS and SESSIONDATA table not shown in our figure 7 are C9 and C10. They specify that the long-integer begin and end time of a session recorded in SESSIONDATA must also occur in the EVENTS table.

We will split up the information contained in the additional fields according to their different meanings and create tables for them. The events which have no additional fields are specified in the EVENTS table and need no additional tables.

The first table with the additional information is the DRUGEVENTS table. This table contains the information on all drug administrations (including the gasses). The set of

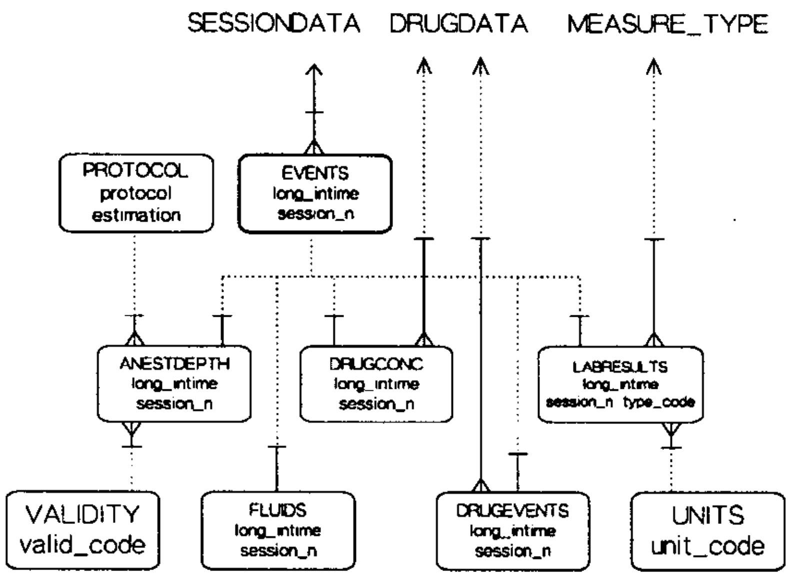
SESSIONDATA
DRUGDATA
MEASURE_TYPE
PROTOCOL protocol estimation
EVENTS long_intime session_n
ANESTDEPTH long_intime session_n
DRUGCONC long_intime session_n
LABRESULTS long_intime session_n type_code
VALIDITY valid_code
FLUIDS long_intime session_n
DRUGEVENTS long_intime session_n
UNITS unit_code

Figure 7 *Datastructure of the events*

attributes *session_n* and *long_intime* is a key within this table and also a foreign key to the EVENTS table, of which the values of these key attributes are a subset (C12). This is also true for the following tables with additional information we will create. The other time attributes and event number are not included because they are already specified in the EVENTS table and not needed in this table (also true for other additional tables). The event description and event type description (redundancy) however, are included so that the information is readable without having to search the EVENTS table; we want to know how which drug was administered. The *correct* attribute (redundancy) is included to avoid mistakes. Otherwise we would have to look into the EVENTS table to make sure that the information is correct.

The attributes that contain the additional information are specified in appendix B. The fact that we have contained all drugevents in one table implies that the *cumdose* and *cum_unit* attributes can have NULL (an empty entry) values in the case of gas administration, for a cumulative dose has no meaning in this case (C1). The units in this table are contained in separate attributes, for the numeric values in this table will be used for calculations. The drug number in the event list is converted to a drug code which is a foreign key to the DRUGDATA table of the subject-session datamodel. Constraint C2 specifies that the cumulative doses of continuously and intermittently administered drugs have to increase in time during a session, if the events have been entered correctly.

The DRUGCONC table contains all drug concentrations of the drugs that have been administered during a monitoring session. Key within this table is of course again *session_n, long_intime*. We remark that the same drug can be administered in different ways at different times with different concentrations. The table is specified in appendix B and needs no further explanation after the extensive explanation of the table DRUGEVENTS.

The FLUIDS table contains the additional information on fluids and the formal specification is given in appendix B. The *eventtype* attribute is left out because the event description supplies the needed information. The *cumamount* of a fluid must increase in time during a session for the events that have been entered correctly (C3). The rest of the table needs no further explanation.

We already stated in chapter 6.2.1 that the estimation of anesthetic depth would end up in a different category and in fact it is now a category by itself. This table will need some further explanation. During the monitoring session the anesthesiologist will give an estimation of the anesthetic depth of the subject according to a previously arranged protocol. This estimation will always be an integer and a protocol will state the meaning of this integer. Thus we introduce the two tables ANEST_DEPTH and PROTOCOL that are specified in appendix B.

First we describe ANEST_DEPTH. The attributes *session_n* and *long_intime* are clear. *Estimation* contains the estimation of anesthetic depth and *protocol* refers to the protocol that has been used. The set of attributes *protocol* and *estimation* is a foreign key to the PROTOCOL table. Because the anesthesiologist can only make an estimation of the anesthetic depth of the subject he will also make a statement about the validity of his estimation. This information is contained in the attribute *valid_code* which is a foreign key and refers to the lookup-table VALIDITY that contains a list of possible statements that can be made about the validity of information. *Valid_code* is the key within this table and the only other attribute of this table is a description of the validity statement. We make use of a lookup table to make sure that only standard validity statements with a clear meaning can be recorded.

The PROTOCOL table describes the protocols that have been used during the monitoring sessions. This table contains the attributes *protocol, estimation* and *description* and for each protocol a description is given for each estimation value of this protocol. The set of attributes *protocol* and *estimation* uniquely identifies a tuple within the PROTOCOL table and the same set of attributes in the ANEST_DEPTH table is therefore a foreign key to this table as we have already stated.

In chapter 6.2.1 we stated that samples could be taken from the fluids that leave the body of the subject and brought to be analyzed in a laboratory. The results of these analyses are also included in our database and thus related to the fluids events. The table LABRESULTS contains the results of the fluid analyses and we have used a special construction to record them. We need this construction to solve the problem that it is impossible to foretell what different kinds of analyses will be performed, which will differ for each fluid type, and what analyzed quantities have been measured. On top of that an analysis of the same fluid, say blood, may be extensive for one monitoring session but very brief for another session. It is therefore impossible to split these analyses up in different tables or contain them in the usual way in one table. We will have to use a special construction.

In this construction we use an attribute to specify what quantity is measured, an attribute to specify the value resulting from the measurement and an attribute to specify the unit of the quantity. To make sure that always the same representations for quantities and units is used we need the lookup table MEASURE_TYPE and UNITS. The attribute presenting the measured quantity is therefore a code (*type_code* which is a foreign key) that refers to the MEASURE_TYPE table and the unit attribute is represented by *unit_code* which is a foreign key to the UNITS table. MEASURE_TYPE contains all possible quantities that can be measured, where *type_code* is the key within this table and description contains a description of the analyzed quantity. The same holds for the UNITS table: *unit_code* is the key within the table and *description* contains a description of the unit. These lookup tables are also needed in the final subject-session model and are specified in the subject-session part of appendix B.

In the formal specification we see that the *value* attribute can only contain numeric values (attribute constraint) so that it can always be used for calculations. It is therefore impossible to insert text (mistakenly or on purpose) into this field. What happens when the analyzed quantity can only have a textual value, for example rhesus factor: + or - ? We can solve this problem by converting the possible textual values to a numeric value. The *unit_code* will then refer to the table UNITS that will give a description of these value(s) and what is actually meant by them.

The other attributes of the LABRESULTS table are *session_n* and *long_intime* (foreign key) which refer to the EVENTS table that specifies which fluid was analyzed. The key within this table is *session_n*, *long_intime* and *type_code*, for it uniquely identifies a measured quantity of a fluid sample taken within a session.

6.3 Equipment and equipment settings

6.3.1 The subject-session datastructure of equipment settings

In section 5.3 we described a datastructure for equipment settings that was included in the subject-session datamodel and we stated that this datastructure contained some serious flaws. The idea behind this datastructure was to record the initial equipment settings at subject-session level and to record changes in settings at timedata level. The equipment and the settings were to be split up in tables according to the different physiologic parameters they recorded during monitoring sessions. So first a list was made up of all possible physiologic parameters that could be recorded during a monitoring session, then for each parameter a list of possible equipment used in relation to the monitoring of this parameter had to be made up with all their adjustable quantities. As seen in figure 5 of section 5.2 this resulted in nine tables, each containing the initial equipment settings for the recording of a specific physiologic parameter. The tables included an equipment name attribute and attributes representing the possible adjustable quantities and containing the initial set value. It was then thought to be possible to list all the adjustable quantities of equipment used for the monitoring of a specific parameter. For example EEG monitoring equipment would have adjustable quantities like gain, filtering etc.

If we take a closer look at this intended datastructure its limitations will immediately become clear: we can't predict what physiologic parameters will be monitored in the future and thus new tables may have to be created in the future that contain the equipment settings of these new parameters. It is also impossible to list all adjustable quantities of equipment used in relation to the monitoring of a parameter. Different adjustable quantities might look the same, such as filter 50-200 Hz and filter 100-250 Hz, but a distinction has to be made between these small differences. The equipment used in relation to the monitoring of a parameter may include totally different devices with totally different adjustable quantities. The inclusion of all these adjustable quantities as attributes in our table would lead to very wide tables and a lot of empty entries for each row that contains an equipment setting.

The retrieval of data from this datastructure presents some problems too. The initial settings and the changes in settings are recorded in different tables and to retrieve an equipment setting valid at some time point will require searching of two tables. The settings of a session are split up over a large amount of tables which is not convenient in the case of data retrieval. If the tables that contain the changes in equipment settings are the same as the initial settings tables, except for one additional long-integer time attribute, then they will contain a lot of empty entries. When only one quantity is adjusted then all the other quantity attributes will be empty.

There still remains one other very important problem which has to be solved: how can we distinguish between two devices that are exactly the same but which have both been used in one monitoring session?

6.3.2 A compact structure for equipment settings

We will now develop a datastructure for equipment adjustments (see figure 8) that does give us the flexibility we need and that will solve the problems presented in the previous chapter. In the subject-session model, equipment is only referred to by name, which is not enough. It is possible that the names of two different devices are the same and we also need a description of the equipment used in monitoring sessions. We therefore create a lookup table EQUIPMENT that contains a list of all the equipment used during monitoring sessions. Each apparatus is assigned with a unique code (*equip_code*) and the other attributes contain the name, the type and a description of the apparatus (see appendix B).

We are now able to refer to the different kinds of equipment that have been used during sessions, but the problem of distinguishing between two of the same devices (the same *equip_code*) used within the same session still exists. We solved this problem by creating a new table EQUIPMENT_USE that specifies the usage of each device. When a specific device is used in a session we assign it with a unique (key) usage number (*use_n*), so two devices with the same *equip_code* will always end up with a different usage number. The attribute *equip_code* in is a foreign key of the EQUIPMENT table and represents the device and we only record equipment in the EQUIPMENT table when it has been used (C13). We will also have to know in what session the equipment was used, so EQUIPMENT_USE contains the attribute *session_n* which is a foreign key to the SESSIONDATA table. Equipment will always be used during a session (C14).

Medical regulations and research purposes require the recording of the special institution code or serial number of the used equipment, through which it will always be possible to retrace an apparatus that has been used in a session and check if there is something wrong with it. The code attached to the apparatus by the institution or when such a code is absent, the serial number, will be contained in the attribute *retrace_code*. The institution where the apparatus is from, is specified by the attribute *instcode* which refers to the INSTITUTIONS table at subject-session level.

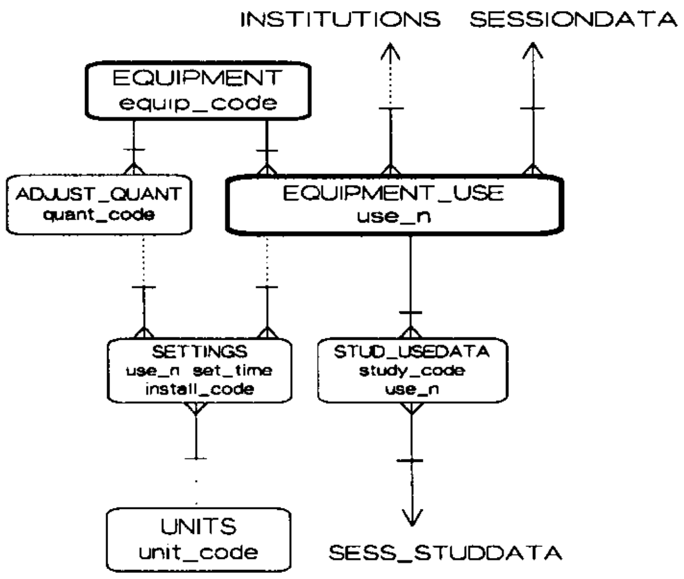

Figure 8 *Datastructure of the equipment settings*

Because of the fact that several studies may be performed during a monitoring session we also want to know what equipment was used for what study. We therefore create the coupling table STUD_USEDATA with attributes *session_n study_code* and *use_n* (see appendix B). *Session_n* is redundant in this table but is convenient for data retrieval purposes. Equipment that has not been used for any study in particular will get a special general study code 'aa-000' reserved for these cases. The set of attributes *session_n* and *study_code* is a foreign key to the SESS_STUDDATA table, but only if *study_code* is not the general code 'aa-000' (C15).

In section 6.3.1 we already stated that it was impossible to make a list of all the adjustable quantities of all equipment that could possibly be used. This problem is yet another version of the problem we had with the recording of laboratory analyses in section 6.2.3 and can thus be solved. We will record all equipment settings in one table, the SETTINGS table. The first attribute of this table specifies the apparatus by its unique usage number *use_n* and the next attribute contains the long-integer time (*set_time*) of the setting. The adjustable quantity is represented by a unique quantity code attribute (*quant_code*) which is a foreign key to the lookup table ADJUST_QUANT that contains a list of all adjustable quantities specified for each apparatus that has been used in the monitoring sessions. *Quant_code* is the key within the ADJUST_QUANT table, *equip_code* represents the apparatus and the attribute description contains a description of the adjustable quantity. The adjustable quantities are specified for each apparatus to avoid confusion about the meaning of quantities of devices that might look the same, but are in fact different.

We now return to the SETTINGS table. The set value of the adjustable quantity is recorded in the attribute value and the matching unit is represented by *unit_code*, which is a foreign key to the units table. The *value* attribute may contain only numeric values and *unit_code* has the same function as described in section 6.2.3 in the LABRESULTS table. The key within SETTINGS is the set of attributes *use_n*, *set_time* and *adjust_code* that uniquely identifies a tuple within this table.

The datastructure developed here is very flexible and can contain a lot of different equipment. We can include drugpumps (DRUGPUMPS table in subject-session model can be omitted) in our EQUIPMENT table and record the connection of a drugpump to catheters in our SETTINGS table. Catheters can be included in EQUIPMENT (*retrace_code* and *instcode* have NULL values) and the site of the catheters can be recorded in SETTINGS. The *value* attribute will always be numeric but the *unit_code* will specify the meaning of the numeric value. We remark that not every apparatus is adjustable (no entries in tables ADJUSTMENT and ADJUST_QUANT) as denoted by the relations between the tables.

We have succeeded in designing a compact and flexible structure for the equipment settings. The disadvantage of this structure lies in its readability. Numeric values are used even when the setting is in fact a text string. We have to search the UNITS table to find the meaning of such a numeric value.

6.4 The montage and electrodes

In this section we will design a part of the datamodel (see figure 9) that is typically characteristic to a neuro-electrophysiologic datamodel. We can safely assume that there will always be a montage of electrodes in neuro-electrophysiologic monitoring sessions (C16). This montage is very important and will often be referred to by users of the database system. It is possible to include the montage of a session in the SETTINGS table by regarding the electrodes as equipment. Because of the fact that the SETTINGS table is difficult to read in the case of non-numeric settings and because the montage is so important and will often be referred to we will create separate tables to record the montage and all related information.

First we create two lookup tables to make sure that users won't invent names and descriptions of their own but will use the standard described in the lookup tables. The first lookup table ELECTRODE_PLACE contains a list of all possible placements of electrodes on a body according to the standard 10-20 system. The attribute *placement* (key) represents a unique placement on the body. The redundant attribute *bodypart* specifies on which part of the body the placement is located and is included for convenient retrieval of electrode placements on a bodypart.

The second lookup table ELECTRODES contains the types of electrodes that might be used during a session. Each type of electrode is assigned with a unique (key) electrode code (*elec_code*) and *elec_type* and *brand* specify the type and the brand of the electrode.

The MONTAGE table contains the actual montage of each monitoring session and which is not liable to changes during a session. The attribute *session_n* (foreign key to SESSIONDATA) specifies the session that the montage belongs to, *elec_code* (foreign key to ELECTRODES) specifies the electrode that has been used, *placement* (foreign key to ELECTRODE_PLACE) specifies the placement of the electrode and *attachment* indicates the way of attachment to the body. A tuple is uniquely identified by *session_n* and *placement* and the complete montage of a session is recorded by all tuples containing that specific session number.

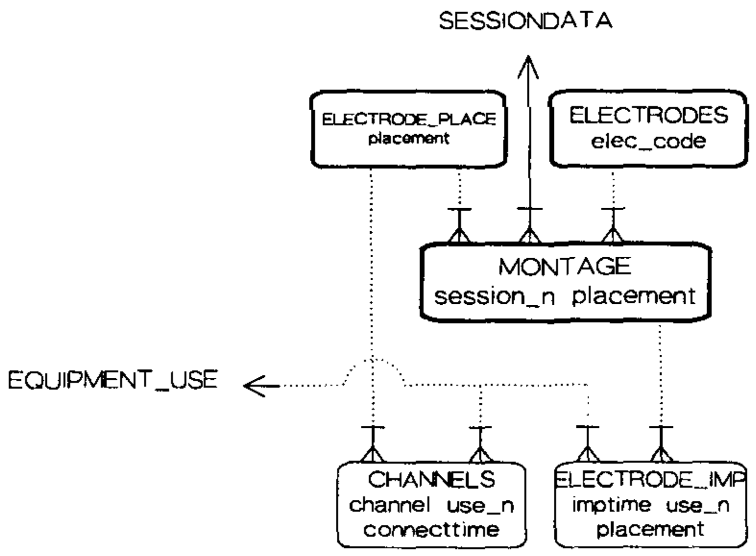
SESSIONDATA
ELECTRODE_PLACE placement
ELECTRODES elec_code
MONTAGE session_n placement
EQUIPMENT_USE
CHANNELS channel use_n connecttime
ELECTRODE_IMP imptime use_n placement

Figure 9 *Datastructure of the electrodes information*

The electrodes of a montage are connected to the input channels of the monitoring equipment and these connections will be recorded in table CHANNELS. The monitoring device to which the electrodes are connected is denoted by *use_n*, the usage number of the device. Each channel of this device has its own number and we will use the attribute *channel* to record this number. The connections can be changed during a session and therefore we add the attribute *connecttime* to specify the long-integer connect time of the electrode. These three attributes are also the key within this table. We can now specify the derivation (uni- or bipolar) and to which electrode places the positive, negative and ground input of the channel are connected. The attributes we add are respectively *uni/bipolar, place_pos, place_neg* and *ground.* In the case of a unipolar derivation the *ground* attribute will contain a NULL (empty) entry (C4). The channel inputs have to be connected to different electrode places (C5, C6, C7).

The electrode impedances are checked during monitoring session to make sure that the impedance is below some preset maximum value so that the parameter measurements are reliable. The measurements of these electrode impedances are contained in table ELECTRODE_IMP. A channel of one of the monitoring devices will be used for this measurement and the attributes *channel* and *use_n* specify the used channel and the monitoring device. The attribute *imptime* is the long-integer time of measurement and we also include the session number (*session_n*) in this table to make data retrieval more convenient. Otherwise we would need *use_n* and the table EQUIPMENT_USE to retrieve the session number, but by including *session_n* we can easily retrieve all the impedances that have been measured during a monitoring session. The attribute *placement* contains the placement of the measured electrode and together with *session_n* it forms a foreign key to the MONTAGE table. The set of attributes *session_n*, *imptime* and *placement* is a key within the ELECTRODE_IMP table and the set *use_n*, *imptime* and *placement* forms an alternative key within this table. The other attributes specify the placement of the ground electrode (*ground*), the measured impedance (*impedance*) and the unit (*unit*). *Session_n* and *ground* form again a foreign key to MONTAGE.

6.5 The measured physiologic parameters

6.5.1 Raw physiologic data

Most of the collected raw physiologic data is not fit for storage in a database because of the enormous amount of samples that have been collected during a monitoring session which would require too much memory to store. This raw data will therefore be stored on other storage facilities. What we do include in our database is information on the collected raw physiologic data and we include the data derived from the raw data after a form of dataprocessing. The storage of derived data will be described in the next section.
The datastructure diagram of this part is given in figure 10.

The information on the collected raw physiologic data we want to store is recorded in table RAW_FYSDATA. The raw data of each physiologic parameter collected during a session is assigned a unique number which is contained in the attribute *raw_n* (key). We will now describe the other attributes of this table one by one.

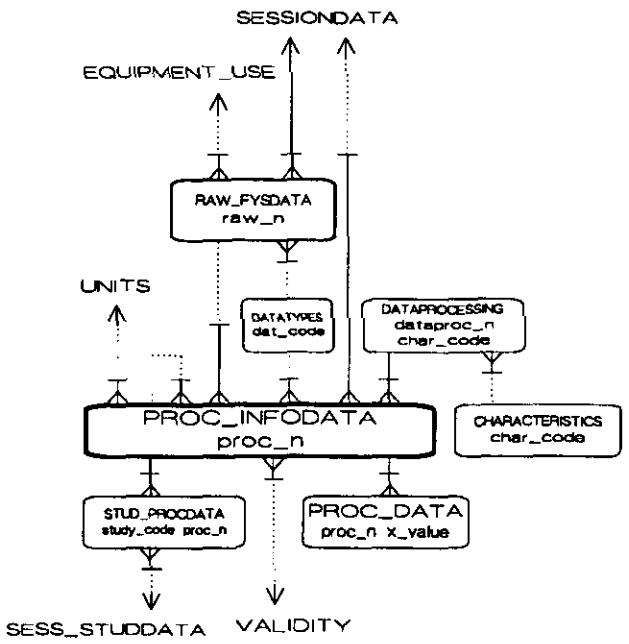

Figure 10 *Datastructure of the collected and derived data*

Dat_code (foreign key) contains a code that refers to the table DATATYPES. It is a unique code within the DATATYPES table and represents what type of data (EEG, ECG etc.) has been monitored. The *description* attribute in the DATATYPES table describes the data type.

Begintime and *endtime* represent the long-integer times of the begin and end time of the monitoring of the physiologic parameter and *use_n* (foreign key to EQUIPMENT_USE) specifies the monitoring device that registered the data. *Sample_freq* represents the sample frequency used by the monitoring device to sample the measured parameter and *unit* contains the unit of the sample frequency. The sample frequency is included (redundant, recorded in SETTINGS) for reasons described in the next section.

The session number (redundant, *use_n* implies a session number) is included again for more convenient data retrieval. *File* refers to the file with the actual stored raw data, *format* specifies the format in which the data is stored and ***medium*** refers to the medium on which the raw data is stored.

6.5.2 Data derived from raw physiologic data

Raw physiologic data is often used as the starting-point in the research on this data. The raw data will often undergo several steps of dataprocessing to visualize the information contained in the raw data. We can picture these steps of dataprocessing by a tree structure as seen in figure 11. The root of the tree represents the original raw data, a tree branch represents some form of dataprocessing and a node represents the derived processed data set. Each interval of the originating data (parent node) can be used for dataprocessing step (branch) and these intervals may well overlap.

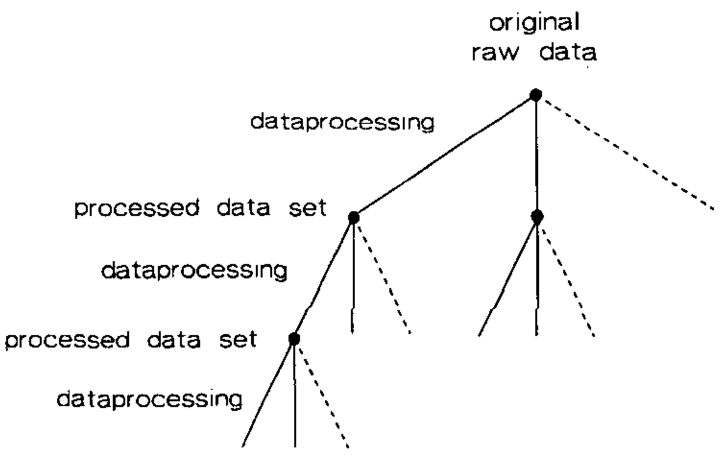

Figure 11 *Tree structure of dataprocessing steps*

Although the storage of processed data sets requires a lot of storage space, we will store the processed data in our database as this is one of the purposes of the database system. Much research will be performed on processed data and the database system has to make this research much easier. The processed data sets can also be used as an input to application programs that visualize the data on screen. The tree structure described above will also be used in our table structure.

First we assign a unique number to each processed data set (*proc_n*). Because of the large amount of processed data we have to store, we need a data structure with economical use of storage space. The table PROC_DATA that contains the actual processed data has therefore only three attributes: the number of the processed data set (*proc_n*), the value of the x-coordinate (*x_value*) and the value of the y-coordinate (*y_value*). Processed data that consists of more than two coordinates should first be converted to a two coordinate representation. The key within this table is the set *proc_n, x_value*. The units of the coordinates are recorded in the PROC_INFODATA table that contains information on the processed data sets.

The processed data sets can now be stored with a minimum of storage space, but we need the PROC_INFODATA table to record the information on these processed data sets such as the information contained in the tree structure of figure 11. *Proc_n* (a node in the tree structure, with exception of the root) is the unique number assigned to a processed data set and is the key within PROC_INFODATA. The attribute *dat_code* is again a foreign key to the DATATYPES table and represents the type of data or curve of the processed data set. The data set that *proc_n* refers to, has been subjected to some kind of dataprocessing and we want to record what dataprocessing has been used. Each type of dataprocessing is assigned a unique number, the dataprocessing number, and the attribute *dataproc_n* records this number which refers to (foreign key) the DATAPROCESSING table.

The problem of recording the different types of dataprocessing is the same as the problem of recording laboratory analyses and equipment settings: we can't predict the types of dataprocessing that will be used and we can't predict their characteristics. We therefore create the same datastructure we used for the other problems. The attribute *dataproc_n* is the number of the processed data set and the key within the DATAPROCESSING table, *char_code* is a code for a characteristic of dataprocessing and *value* specifies the value of the characterisation field. We remark that *value* may contain numeric values as well as strings and that a possible unit will also be recorded in the value field. The values in this table will not be used for calculations and are therefore allowed to have other values besides numeric values.

The *char_code* attribute of the DATAPROCESSING table is a foreign key to the lookup table CHARACTERISTICS and is a key within this last table. The only other attribute (*description*) of CHARACTERISTICS contains a description of the characterisation field.

Returning to our PROC_INFODATA table we will describe some other attributes that contain information we need on the processed data. *Valid_code* is a foreign key to the VALIDITY table and represents a statement on the validity of the processed data set. The processed data may for example be unreliable because of artifacts that took place during that measurement interval. *X_unitcode* and *y_unitcode* specify the units of the x-coordinate and y-coordinate respectively of the *x_value* and *y_value* of the PROC_DATA table. *Session_n* is again included for more convenient data retrieval.

The only information that still has to be stored in the PROC_INFODATA table is what originating data and what interval of this originating data has been used for dataprocessing. There are two possibilities: the originating data is raw data or the originating data is processed data (the parent nodes in the tree structure). We add the attributes *raw_n* (foreign key to RAW_FYSDATA) to refer to the number of the original data (the root) and *inproc_n* to refer to the number of the originating data (a node, with exception of the root). So *inproc_n* contains a number that must also occur in another tuple of this table, but here as the number of a processed data set (this *proc_n* is parent node). If the originating data is raw data (the root) then *inproc_n* will contain a NULL (empty) entry and we will thus know that the originating was raw data. We remark that the relation between *inproc_n* and *proc_n* (C8) is called an internal subset requirement (see section 3.6).

The interval of the originating data that has been used for dataprocessing is specified by the attributes *begin_int* and *end_int* that represent the begin and the end of the interval respectively. They contain a numeric value and the unit of this value is recorded in the tuple specified by *inproc_n* or when NULL, by *raw_n*. The sample frequency in the RAW_FYSDATA table is used to extract the time unit. The interval that has been used for dataprocessing consists of values that can be converted to long-integer times which specify the time interval of the collected raw data that was needed for this dataprocessing (several steps of dataprocessing may have been used). This time interval is included so that we can easily retrieve the events (especially artifacts) that occurred during this time interval. The attributes that contain the long-integer begin and end time of the interval are respectively *begintime* and *endtime*.

The last table of our datamodel we will discuss is the STUD_PROCDATA table. This coupling table specifies what processed data set belongs to what study. The attribute *proc_n* (foreign key to PROC_INFODATA) specifies the number of the processed data set, *study_code* and *session_n* are the familiar study code and session number. *Session_n* is again redundant but convenient for data retrieval purposes.

6.6 The complete time-data model

In the previous sections of this chapter we have described the time-datamodel step by step. We can now give a complete diagram of the datastructure of the time related information (see figure 12). As we have already mentioned this diagram also contains some tables that actually belong at subject-session level but were created during the development of the time-datamodel.

The attribute constraints recorded in the formal specification of the time-datamodel are still too global but will be worked out further in the near future. First it has to be decided what coding will be used for the various codes in our model and secondly, a lot of these constraints

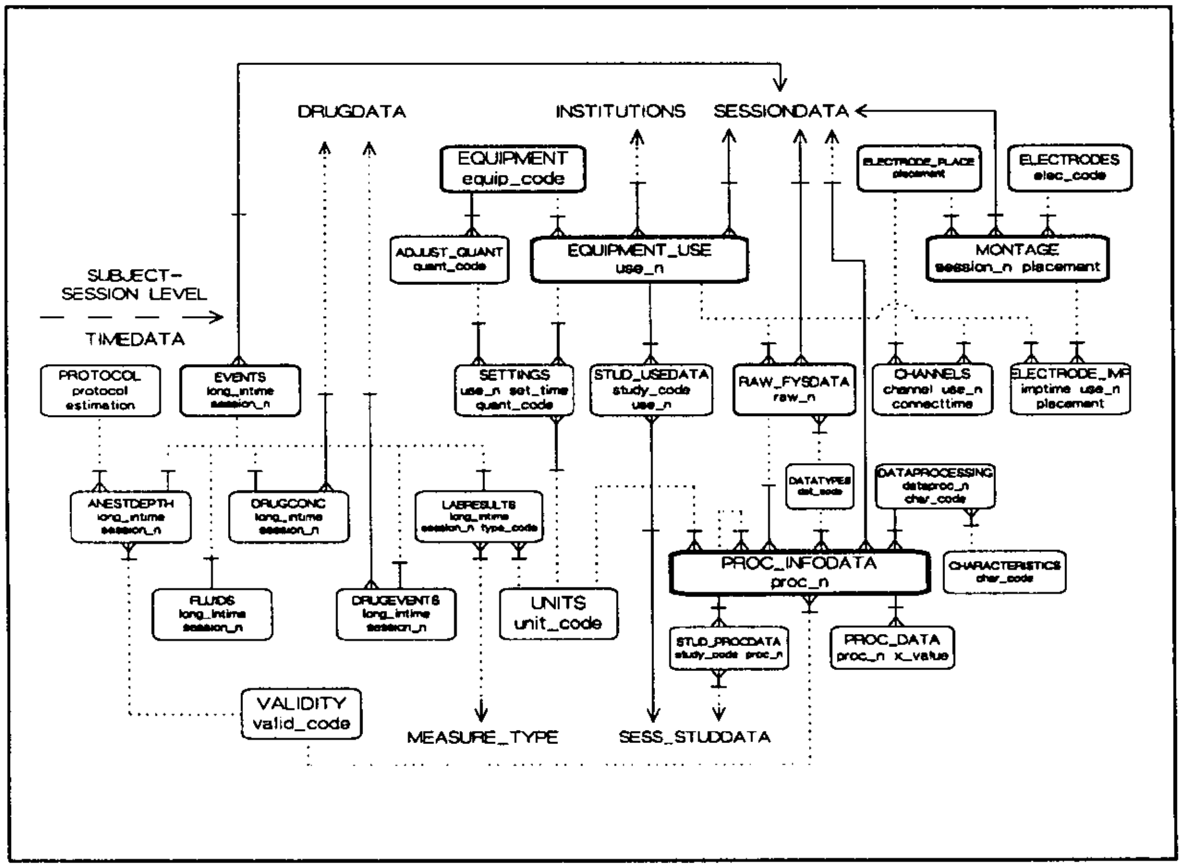
DRUGDATA
INSTITUTIONS
SESSIONDATA
EQUIPMENT equip_code
ELECTRODE_PLACE placement
ELECTRODES elec_code
ADJUST_QUANT quant_code
EQUIPMENT_USE use_n
MONTAGE session_n placement
SUBJECT-SESSION LEVEL
TIMEDATA
PROTOCOL protocol estimation
EVENTS long_intime session_n
SETTINGS use_n set_time quant_code
STUD_USEDATA study_code use_n
RAW_FYSDATA raw_n
CHANNELS channel use_n connecttime
ELECTRODE_MP imptime use_n placement
DATATYPES dat_code
DATAPROCESSING dataproc_n char_code
ANESTDEPTH long_intime session_n
DRUGCONC long_intime session_n
LABRESULTS long_intime session_n type_code
PROC_INFODATA proc_n
CHARACTERISTICS char_code
FLUIDS long_intime session_n
DRUGEVENTS long_intime session_n
UNITS unit_code
STUD_PROCDATA study_code proc_n
PROC_DATA proc_n x_value
VALIDITY valid_code
MEASURE_TYPE
SESS_STUDDATA

Figure 12 *The datastructure of the time-datamodel*

will depend upon the DBMS which is used and upon the user-interface . It is for example not possible to show long lines of text on the users screen and the attribute constraints should specify this. In the subject-session model that is also included in appendix B and that has been designed by H. Kuipers we can see a further specification of these constraints. The database constraints specified in appendix B represent the relations between the tables also visualized in the diagrams of the datastructures.

6.7 The complete datamodel of EMDABS

In figure 13 we have coupled the diagram of the time-datamodel to the final subject-session diagram which gives us a survey of the complete datastructure of EMDABS. The table boxes contain the abbreviated table indices as specified in the database characterisation of appendix B that contains the total formal specification. The subject-session datamodel contains some changes with respect to the datamodel described in paragraph 5 which we will now discuss.

The tables PRE_SESS_GENERAL, PRE_SESS_BLDSERUM, PRE_SESS_HEMATOL and PRE_SESS_URINE have been combined to one PRE_SESS_GENERAL table which has the same construction as the LABRESULTS table. This construction is more flexible because we can include any type of pre-session information. A time field that states the registration time is also included in this table.

The table PRE_SESS_DRUGDATA has been changed to PRE_MEDICATION with the addition of one time attribute that contains the administration time.

The tables SESS_PERSONNELDATA and SESS_PERS_STUDDATA have been combined to one STUD_PERSONNEL table because of the introduction of the special 'aa-000' study_code that states that the information is not study related but general.

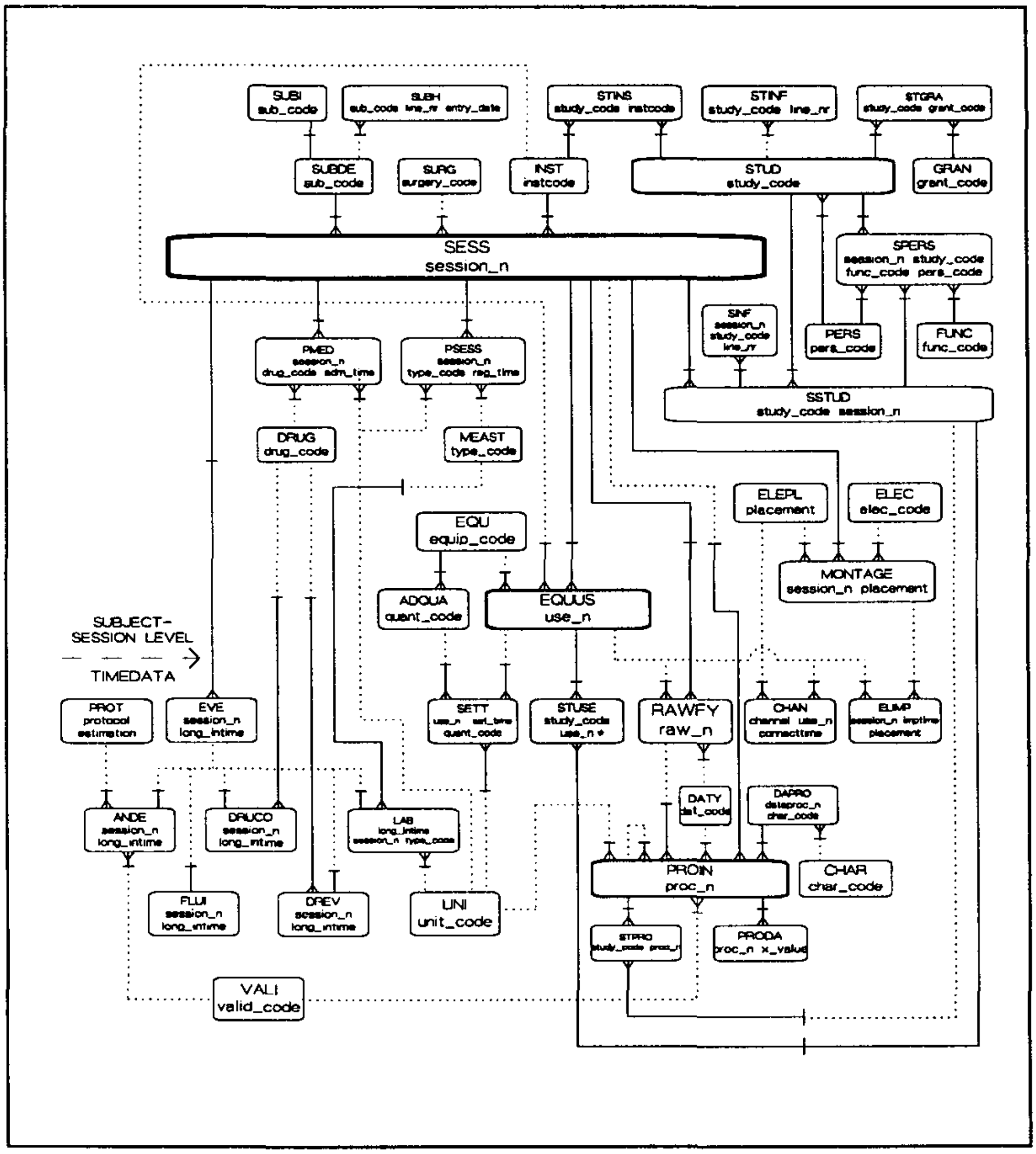
SUBI sub_code
SUBH sub_code line_nr entry_date
STINS study_code instcode
STINF study_code line_nr
STGRA study_code grant_code
SUBDE sub_code
SURG surgery_code
INST instcode
STUD study_code
GRAN grant_code
SESS session_n
SPERS session_n study_code func_code pers_code
SINF session_n study_code line_nr
PERS pers_code
FUNC func_code
PMED session_n drug_code adm_time
PSESS session_n type_code reg_time
SSTUD study_code session_n
DRUG drug_code
MEAST type_code
EQU equip_code
ELEPL placement
ELEC elec_code
MONTAGE session_n placement
ADQUA quant_code
EQUUS use_n
SUBJECT-SESSION LEVEL
TIMEDATA
PROT protocol estimation
EVE session_n long_intime
SETT use_n set_time quant_code
STUSE study_code use_n *
RAWFY raw_n
CHAN channel use_n connecttime
ELIMP session_n imptime placement
ANDE session_n long_intime
DRUCO session_n long_intime
LAB long_intime session_n type_code
DATY dat_code
DAPRO dataproc_n char_code
PROIN proc_n
CHAR char_code
FLUI session_n long_intime
DREV session_n long_intime
UNI unit_code
STPRO study_code proc_n
PRODA proc_n x_value
VALI valid_code

Figure 13 *The complete datastructure of the EMDABS datamodel*

7 CONCLUSIONS AND FUTURE WORK

In this report we have described the development of a datamodel of the time related information that will be stored in EMDABS. The datamodel is flexible in the sense that it can be used for all kinds of neuro-electrophysiologic research and probably even for other types of clinical research. All the time-related information needed for this research can be recorded in the time-datamodel and the retrieval of important information that is used often for research purposes is possible without much effort. Five years after the start of the EMDABS project there now exists a complete and good datamodel exactly specified by a convenient and elegant formal mathematical technique.

This formal technique has proven to be very useful for the specification of the semantics of the datamodel in the form of constraints. The attribute constraints however, will have to be defined more strictly when more is known about the coding of several code attributes and the facilities of the DBMS. A DBMS has few facilities to record the attribute constraints at database level and a lot of time will be needed to write an application program that enforces these constraints. It is possible that the large amount of information stored in the database and the amount of constraints that has to be checked before data is stored may slow down the system considerably although hardware facilities are still improving rapidly as well as DBMS facilities.

The writing of a user-interface application program will take a lot of time and effort but a lot of application programs written for other database systems already exist and it should be possible to profit from the available experience and knowledge on this subject. We refer to [Yao, 1989; Lutz, 1986] which might prove useful in the further development of the database system.

The data that is collected and recorded by the ERDA system during the monitoring sessions will have to be converted to a format that can be used by the data loader of Oracle.
A program that provides this conversion will be written shortly after this report. The ERDA

system itself will also have to be changed and adjusted to the datamodel. It is easily possible to implement the ERDA system in such a way that certain constraints on the data are already enforced by this system so that EMDABS can waive the checking of these constraints.

This datamodel provides only for the storage of information used for research and not in the storage of research results (often statistical information). The research results can be stored in a separate module that may be designed later and added to the existing datamodel without major changes. The design of a module with research results however, has a low priority.

The standard form of information storage is now specified by this datamodel. The institutions that will share information have to make use of this standard form of data storage otherwise sharing of information will still present problems.

Additional information on the EMDABS project is available through P.J.M. Cluitmans of the Division of Medical Electrical Engineering at the Eindhoven University of technology.

APPENDIX A : MATHEMATICAL TERMS

Some mathematical terms are defined which are used in the formal specification of semantic relational database concepts. The definitions may be somewhat different from the definitions that are usually known for these terms.

First some notations:

Z	the set of whole numbers.
$\mathbb{R}$	the set of real numbers.
$x \in A$	x is an element of set A.
$\mathbf{N}$	$= \{ t \mid t \in Z \wedge t \geq 0 \}$.
$\varnothing$	the empty set.
$\forall_{x \in A}$:	for each element of A holds:.
$\exists_{x \in A}$:	there exists an element x of A for which holds:.
$\Rightarrow$	implies.
$\Leftrightarrow$	if and only if.
$\wedge$	and.
$\vee$	or.
$A \subseteq B$	$\Leftrightarrow \forall_{x \in A} : x \in B$.
$A \cup B$	$= \{ x \mid x \in A \vee x \in B \}$.
$A \cap B$	$= \{ x \mid x \in A \wedge x \in B \}$.
chs(n)	if $n \in \mathbf{N}$ then chs(n) denotes the set of all rows of signs (=strings) that exist of n signs maximally.
[m..n]	$= \{ x \mid x \in Z \wedge m \leq x \leq n \}$.
vnc(n)	if $n \in \mathbf{N}$ then vnc(n) denotes the set of all natural numbers that exist of exactly n numbers (number $\in$ [0..9]).
vng(n)	$= [10^{n-1}..10^{n} - 1]$.
int(n)	$= [-(10^{n} - 1)..10^{n} - 1]$.
NULL	an empty entry in a table.

We will assume the term ordered pair to be known. Notation: (x;y). Let p be an ordered pair then we will denote the first and second coordinate of p respectively as k_1p and k_2p. Thus $p=(x;y) \Rightarrow k_1p=x \wedge k_2p=y$.

We will define the well-known term function in a way that is very convenient to our purpose.

Definition :

f is a function $\Leftrightarrow$ f is a set of ordered pairs and $\forall_{p,q \in f} : k_1p \neq k_1q \vee p=q$.

Example: f = {(nr;27), (name; 'Smith')

The domain and range of f is respectively denoted by dom(f) and rng(f).

Definition :

$\mathrm{dom}(f) = \{ k_1p \mid p \in f \}$ and $\mathrm{rng}(f) = \{ k_2p \mid p \in f \}$.

Definition :

f is a function over A $\Leftrightarrow$ f is a function and dom(f)=A.

Let f be a function and X be a set. The restriction of f is denoted by $f \upharpoonright X$.

Definition : $f \upharpoonright X = \{ p \in f \mid k_1p \in X \}$.

Let V be a set of functions and X be a set. The projection of V on X is denoted by $V \Vert\!\upharpoonright X$.

Definition : $V \Vert\!\upharpoonright X = \{ f \upharpoonright X \mid f \in V \}$.

Let f and g be functions and A be a set.

Definitions :

f and g agree upon A $\Leftrightarrow$ $f \upharpoonright A = g \upharpoonright A$.

f and g are joinable $\Leftrightarrow$ f and g agree upon $\mathrm{dom}(f) \cap \mathrm{dom}(f)$ ($\Leftrightarrow$ $f \cup g$ is a function).

The composition of functions f and g is denoted by $f \circ g$.

Definition : $f \circ g = \{ (x ; f(g(x))) \mid x \in dom(g) \wedge g(x) \in dom(f) \}$.

Let V be a set of functions and h be a function then we define a rename function or attribute transformation h as:

Definition : $V \infty h = \{ f \circ h \mid f \in V \}$.

Definitions :

F is a set function $\Leftrightarrow$ F is a function and $\forall_{p \in F} : k_2 p$ is a set.

F is a regular set function $\Leftrightarrow$ F is a set function and $F \neq \varnothing$ and $\forall_{p \in F} : k_2 p \neq \varnothing$.

Let F be a set function then ΠF denotes the generalised product of F, so:

$\Pi F = \{ f \mid f$ is a function and $\forall_{p \in f} : k_2 p \in F(k_1 p)\}$.

APPENDIX B : THE FORMAL SPECIFICATION OF THE DATAMODEL

```
type  nr = [1..100000];
      code_1 = [10011110000..19911119999];
      date = [19000101..20991231];              year-month-day
      ss_code = [vnc(3)-vnc(2)-vnc(4)];
      ltime = vnc(9);                           long-integer-time
      code_2 = ['aa-01-00'..'zz-12-31'];        personnel code
      time = ['00-00-00'..'24-59-59'];          normal time form
      body = {'head','neck','breast','arm','leg'};
      drg = {'µg/hour','mg/hour','ml/hour','µg','mg','ml','%'};
      con = {'mmol/l','mol/l','µg/l','mg/l'};
      freq = {'Hz','KHz','MHz'};
      imp = {'Ω','KΩ','MΩ'};
      pos = { t | t ∈ R ∧ t≥0 };

      tatp subj_iddata =                        subject identification
       tutp T-subi =                            begin of subject-session model
        obcar F-subi =
           sub_code     : code_1;               subject code
           entrydate    : date;                 datum of entry
           last_name    : chs(15);              last name
           first_name   : chs(10);              first name
           initials     : chs(3);               initials
           ss_number    : ss_code;              social security number
           med_record   : ['a00000'..'z99999']; medical registration nr.
        endobcar;
       endtutp;
       keys {{sub_code}}
      endtatp;
```

```
tatp subj_demogrdata =                                  demographic information
 tutp T-subd =
  obcar F-subd =
      sub_code       : code_1;                          subject code
      entry_date     : date;                            date of entry
      type           : chs(15);                         subj-type: human or animal
      date_birth     : date;                            date of birth
      sex_mf         : {'m','w'};                       sex
      handed_rl      : {'r','l','rl','lr'};             left- right handed
      bldgroup       : {'O','A','B','AB'};              blood type
      rhesusfac      : {'+','-'};                       rhesus factor
  endobcar;
  tuc t(entry_date) ≥ t(date_birth) ∧
      (t(type) ≠ 'human') ⇒ t(handed_rl) = NULL
 endtutp;
 keys {{sub_code}}
endtatp;


tatp subj_histdata =                                    additional medical information
 tutp T-subh =
  obcar F-subh =
      sub_code       : code_1;                          subject code
      line_nr        : vnc(2);                          line number
      entry_date     : date;                            date of entry
      info           : chs(72);                         text line
  endobcar;
 endtutp;
 keys {{sub_code,line_nr,entry_date}}
endtatp;
```

```
tatp surgerydata =                                      operation types
 tutp T-surg =                                          (mutiple types as well)
  obcar F-surg =
      surgery_code  : ['aaa00'..'zzz99'];               operation code
      description   : chs(72);                          operation description
  endobcar;
 endtutp;
 keys {{surgery_code}}
endtatp;

tatp sessiondata =                                      session information
 tutp T-sess =
  obcar F-sess =
      session_n     : nr;                               session number
      sub_code      : code_1;                           subject code
      surgery_code  : ['aaa00'..'zzz99'];               operation code
      instcode      : [chs(3)-chs(3)];                  institution code
      start_date    : date;                             start date session
      end_date      : date;                             end date session
      start_time    : ltime;                            begintijdstip session
      end_time      : ltime;                            eindtijdstip session
  endobcar;
  tuc t(start_date) ≤ t(end_date) ∧ t(start_time) < t(end_time)
 endtutp;
 keys {{session_n}}
endtatp;
```

```
tatp institutions =                                  institution information
 tutp T-inst =
  obcar F-inst =
      instcode      : [chs(3)-chs(3)];               institution code
      instname      : chs(30);                       institution name
      instplace     : chs(30);                       location of institution
  endobcar;
 endtutp;
 keys {{instcode}}
endtatp;

tatp studydata =                                     study information
 tutp T-stud =
  obcar F-stud =
      study_code    : ['aa-000'..'zz-999'];          study code
      entry_date    : date;                          date of entry
      studytitle    : chs(125);                      name of study
      proj_name     : chs(72);                       name of overall organisation
      pr_researcher : code_2;                        personnel code of 1st research manager
      se_researcher : code_2;                        personnel code of 2nd research manager
  endobcar;
  tuc (t(study_code) = 'aa-000') ⇒ t(entry_date) = NULL ∧ t(studytitle) = 'general'
 endtutp;
 keys {{study_code}}
endtatp;
```

```
tatp grants =                                        grants information
 tutp T-gran =
  obcar F-gran =
      grant_code    : ['aaa-00'..'zzz-99'];  grant code
      grant_inst    : chs(72);               providing institution
      grant_inst_nr : chs(40);               grants disposal number
      title         : chs(72);               name of the grant
  endobcar;
 endtutp;
 keys {{grant_code}}
endtatp;

tatp stud_infodata =                                 general study information
 tutp T-stinf =
  obcar F-stinf =
      study_code    : ['aa-001'..'zz-999'];  study code
      line_nr       : vnc(2);                rule number
      info          : chs(72);               text line
  endobcar;
 endtutp;
 keys {{study_code,line_nr}}
endtatp;

tatp stud_grantdata =                                grant dividing information
 tutp T-stgra =
  obcar F-stgra =
      study_code    : ['aa-001'..'zz-999'];  study code
      grant_code    : ['aaa-00'..'zzz-99'];  grant code
  endobcar;
 endtutp;
 keys {{study_code,grant_code}}
endtatp;
```

```
tatp stud_instdata =                            which study on which institution
 tutp T-stins =
  obcar F-stins =
      study_code     : ['aa-001'..'zz-999'];    study code
      instcode       : [chs(3)-chs(3)];         institution code
  endobcar;
 endtutp;
 keys {{study_code,instcode}}
endtatp;

tatp personnel =                                personnel information
 tutp T-pers =
  obcar F-pers =
      pers_code      : code_2;                  personnel code
      name           : chs(35);                 name
      instcode       : [chs(3)-chs(3)];         institution code
  endobcar;
 endtutp;
 keys {{pers_code}}
endtatp;

tatp functions =                                function information
 tutp T-func =
  obcar F-func =
      func_code      : ['aa00'..'zz99'];        function code
      func_name      : chs(35);                 function name
      description    : chs(80):                 function description
  endobcar;
 endtutp;
 keys {{func_code}}
endtatp;
```

```
tatp sess_infodata =                              general session information
 tutp T-sinf =
  obcar F-sinf =
      study_code    : ['aa-000'..'zz-999'];  study code
      session_n     : nr;                    session number
      line_nr       : vnc(2);                line number
      keyword       : chs(20);               keyword to text line
      info          : chs(72);               text line
  endobcar;
 endtutp;
 keys {{session_nr,study_code,line_nr}}
endtatp;

tatp sess_personneldata =                         personnel attending a session
 tutp T-spers =
  obcar F-spers =
      study_code    : ['aa-000'..'zz-999'];  study code
      session_n     : nr;                    session number
      func_code     : ['aa00'..'zz99'];      function code
      pers_code     : code_2;                personnel code
  endobcar;
 endtutp;
 keys {{session_nr,study_code,func_code,pers_code}}
endtatp;
```

```
tatp pre_medication =                                pre-operative medication
 tutp T-pmed =
  obcar F-pmed =
      session_n      : nr;                          session number
      drug_code      : ['a000'..'z999'];            drug code
      long_intime    : ltime;                       long-integer administration time
      adm_date       : date;                        date of administration
      adm_time       : time;                        time of administration
      med_type       : chs(15);                     way of administration
      dose           : pos;                         administered dose
      unit           : drg;                         unit of dose
      cumdose        : pos;                         cumulative dose till now
      cum_unit       : {'µg','mg','g'};             unit
  endobcar;
  tuc (t(med_type) = 'continuously') ⇒ t(unit) = 'µg/hour' ∧
      (t(med_type) = 'intermittently') ⇒ t(unit) ∈ {'µg','mg'} ∧
      (t(med_type) = 'gas') ⇒ t(unit) = '%' ∧ t(cumdose) = NULL ∧
      t(cum_unit) = NULL
 endtutp;
tac $\forall_{t,s \in D}$ : ( t(long_intime) > s(long_intime) ∧ t ↾ {med_type, drug_code} =
      s ↾ {med_type, drug_code} ) ⇒ t(cumdose) > s(cumdose)
 keys {{session_n,drug_code,long_intime}}
endtatp;
```

```
tatp drugdata =                                    drug information
 tutp T-drug =
  obcar F-drug =
     drug_code     : ['a000'..'z999'];             drug code
     generic_n     : chs(26);                      generic name
     trade_n1      : chs(16);                      1st brand name
     trade_n2      : chs(16);                      2de brand name
  endobcar;
 endtutp;
 keys {{drug_code}}
endtatp;

tatp pre_sessdata =                                pre-operative information (non-medical)
 tutp T-psess =
  obcar F-psess =
     session_n     : nr;                           session number
     type_code     : ['aaaa00'..'zzzz99'];         code for measured quantity
     long_intime   : ltime;                        long-integer registration time
     adm_date      : date;                         date of registration
     adm_time      : time;                         registration time
     value         : real;                         value of measured quantity
     unit_code     : chs(10);                      unit code
  endobcar;
 endtutp;
 keys {{session_n,type_code,long_intime}}
endtatp;
```

```
tatp sess_studdata =                              which session for which study
 tutp T-sstud =
  obcar F-sstud =
      study_code    : ['aa-001'..'zz-999'];       study code
      session_n     : nr;                         session number
  endobcar;
 endtutp;
 keys {{study_code,session_n}}
endtatp;

tatp measure_type =                               possible measurable quantities
 tutp T-meast =                                   (section 6.2.3)
  obcar F-meast =
      type_code     : chs(10);                    code for measured quantity
      description   : chs(72);                    description
  endobcar;
 endtutp;
 keys {{type_code}}
endtatp;

tatp units =                                      units
 tutp T-uni =                                     (section 6.2.3)
  obcar F-uni =
      unit_code     : chs(10);                    code for unit
      description   : chs(72);                    description
  endobcar;
 endtutp;
 keys {{unit_code}}
endtatp;
```

```
tatp events =                             contains eventlist  begin of time-datamodel
 tutp T-eve =                             (section 6.2.3)
  obcar F-eve =
     session_n     : nr;                  session number
     long_intime   : ltime;               long-integer time of entry event
     intime        : time;                entry time event
     long_evtime   : ltime;               long-integer time of event
     evtime        : time;                time of event
     ev_number     : nr;                  event number
     event         : chs(72);             description event
     event_type    : chs(72);             description eventtype
     correct       : {'y','n'};           event correctly entered (='y') or not (='n')
  endobcar;
  tuc t(long_intime) ≥ t(long_evtime) ∧ t(intime) ≥ t(evtime)
 endtutp;
 keys {{session_n,long_intime}}
endtatp;
```

```
tatp drugevents =                              drugevents
  tutp T-drev =                                (section 6.2.3)
    obcar  F-drev =
        session_n       : nr;                  session number
        long_intime     : ltime;               long-integer time of entry event
        event_type      : chs(72);             drug administration
        event           : chs(72);             name of drug
        drug_code       : ['a000'..'z999'];    drugcode
        dose            : pos;                 administered dose
        unit            : drg;                 unit of dose
        cumdose         : pos;                 cumulative dose
        cum_unit        : {'µg','mg','g'};     unit of cum_dose
        correct         : {'y','n'};           event correctly entered (='y') or not (='n')
    endobcar;
    tuc t(correct)='y' ⇒ (t(dose)≥0 ∧ t(cumdose)≥0);
        t(event_type)='continuously' ⇒ t(unit)='µg/hour';
        t(event_type)='intermittently' ⇒ t(unit) ∈ {'µg','mg'};
C1      t(event_type)='gas' ⇒ (t(unit)='%' ∧ t(cumdose)=NULL ∧
        t(cum_unit)=NULL)
  endtutp;
C2  tac ∀t,s ∈ D : (t(correct)='y' ∧ s(correct)='y' ∧ t(session_n)=s(session_n) ∧
        t(long_intime) > s(long_intime) ∧ t↾{event_type,event}=s↾{event_type,event})
        ∧ t(event_type)≠'gas' ⇒ t(cumdose) > s(cumdose)
  keys {{session_n,long_intime}}
endtatp;
```

```
tatp drugconc =                                concentrations of administered drugs
  tutp T-druco =                               (section 6.2.3)
    obcar F-druco =
        session_n     : nr;                    session number
        long_intime   : ltime;                 long-integer time of entry event
        event         : chs(72);               name of drug
        drug_code     : ['a000'..'z999'];      drugcode
        eventtype     : chs(72);               drug administration
        concentration : pos\{0};               concentration
        unit          : con;                   unit
        correct       : {'y','n'};             event correctly entered (='y') or not (='n')
    endobcar;
  endtutp;
  keys {{session_n,long_intime}}
endtatp;
```

```
tatp fluids =                                  fluid events
  tutp T-flui =                                (section 6.2.3)
    obcar F-flui =
        session_n      : nr;                   session number
        long_intime    : ltime;                long-integer time of entry event
        event          : chs(72);              fluid event
        amount         : pos;                  administered/liberated amount
        unit           : {'ml'};               unit of amount
        cumamount      : pos;                  cumulative amount
        cum_unit       : {'ml','l'};           unit of cumamount
        correct        : {'y','n'};            event correctly entered (='y') or not (='n')
    endobcar;
    tuc t(correct)='y' ⇒ t(cumamount) ≥ t(amount)
  endtutp;
C3  tac ∀t,s ∈ D : (t(correct='y' ∧ s(correct)='y' ∧ t(session_n)=s(session_n) ∧
        t(long_intime) > s(long_intime) ∧ t(event)=s(event)) ⇒
        t(cumamount) > s(cumamount)
  keys {{session_n,long_intime}}
endtatp;
```

```
tatp labresults =                               results of laboratory analyses of fluids
  tutp T-lab =                                  (section 6.2.3)
    obcar F-lab =
        session_n      : nr;                    session_n
        long_intime    : ltime;                 long-integer time of entry event
        type_code      : chs(10);               code for measured quantity
        value          : R;                     value
        unit_code      : chs(10);               unit code
    endobcar;
  endtutp;
  keys {{session_n,long_intime,type_code}}
endtatp;

tatp anest_depth =                              estimation of anesthetic depth
  tutp T-ande =                                 (section 6.2.3)
    obcar F-ande =
        session_n      : nr;                    session number
        long_intime    : ltime;                 long-integer time of entry event
        estimation     : N;                     estimation of anesthetic depth
        protocol       : chs(72);               used protocol
        valid_code     : chs(10);               code for validity estimation
        correct        : {'y','n'};             event correctly entered (='y') or not (='n')
    endobcar;
  endtutp;
  keys {{session_n,long_intime}}
endtatp;
```

```
tatp validity =                              validity statements on data
  tutp T-vali =                              (section 6.2.3)
    obcar T-vali =
        valid_code    : chs(10);             code for validity statement
        description   : chs(72);             statement description
    endobcar;
  endtutp;
  keys {{valid_code}}
endtatp;

tatp protocol                                anesthesia protocol
  tutp T-prot =                              (section 6.2.3)
    obcar F-prot =
        protocol      : chs(72);             anesthetic depth protocol
        estimation    : N;                   estimation of anesthetic depth
        description   : chs(72);             description
    endobcar;
  endtutp;
  keys {{protocol,estimation}}
endtatp;
```

```
tatp equipment =                               equipment
  tutp T-equ =                                 (section 6.3.2)
    obcar F-equ =
        equip_code    : chs(10);               equipment code
        equip_name    : chs(72);               equipment name
        equip_type    : chs(72);               type of equipment
        description   : chs(72);               equipment description
    endobcar;
  endtutp;
  keys {{equip_code}}
endtatp;

tatp equipment_use =                           equipment usage
  tutp T-equus =                               (section 6.3.2)
    obcar F-equus =
        use_n         : nr;                    usage number
        session_n     : nr;                    session number
        equip_code    : chs(10);               equipment code
        retrace_code  : chs(72);               equipment code of institution or serial number
        instcode      : [chs(3)-chs(3)];       institution code
    endobcar;
  endtutp;
  keys {{use_n},{session_n,retrace_code,instcode}}
endtatp;
```

```
tatp stud_usedata =                              what equipment belongs to what study
  tutp T-stuse =                                 (section 6.3.2)
    obcar F-stuse =
        study_code    : ['aa-000'-'zz-999'];    study code
        session_n     : nr;                      session number
        use_n         : nr;                      usage number
    endobcar;
  endtutp;
  keys {{study_code,use_n}}
endtatp;

tatp adjust_quant =                              adjustable equipment quantities
  tutp T-adqua =                                 (session 6.3.2)
    obcar F-adqua =
        install_code  : chs(10);                 code for adjustable quantity
        description   : chs(72);                 description of adjustable quantity
        equip_code    : chs(10);                 equipment code
    endobcar;
  endtutp;
  keys {{install_code}}
endtatp;
```

```
tatp settings =                                 equipment settingss
  tutp T-sett =                                 (section 6.3.2)
    obcar F-sett =
        use_n           : nr;                   usage number of equipment
        set_time        : ltime;                long-integer time of setting
        quant_code      : chs(10);              code for adjustable quantity
        value           : R;                    set value
        unit_code       : chs(10)               unit code
    endobcar;
  endtutp;
  keys {{use_n,set_time,quant_code}}
endtatp;

tatp electrode_place =                          possible placements of an electrode
  tutp T-elepl =                                (section 6.4)
    obcar F-elepl =
        placement       : chs(72);              placement of electrode
        bodypart        : body;                 bodypart on which electrode is attached
    endobcar;
  endtutp;
  keys {{placement}}
endtatp;
```

```
tatp electrodes =                          electrodes
  tutp T-elec =                            (section 6.4)
    obcar F-elec =
        elec_code     : chs(10);           electrode code
        elec_type     : chs(72);           description of electrode type
        brand         : chs(72);           brand of electrode
    endobcar;
  endtutp;
  keys {{elec_code}}
endtatp;

tatp montage =                             plaments of electrodes within a session
  tutp T-mont =                            (section 6.4)
    obcar F-mont =
        session_n     : nr;                session number
        elec_code     : chs(10);           electrode code
        placement     : chs(72);           placement of electrode
        attachment    : chs(72);           way of attachment
    endobcar;
  endtutp;
  keys {{session_n,placement}}
endtatp;
```

```
    tatp channels =                                  channels (amplifiers)
      tutp T-chan =                                  (section 6.4)
        obcar F-chan =
            channel       : nr;                      channel number
            use_n         : nr;                      usage number of equipment
            connecttime   : ltime;                   long-integertime of connection
            uni/bipolair  : {'uni','bi'};            uni- or bipolar derivation
            place_pos     : chs(72);                 placement of electrode +
            place_neg     : chs(72);                 placement of electrode -
            ground        : chs(72);                 placement of neutral electrode (just bipolar)
        endobcar;
C4      tuc t(uni/bipolair)='uni' ⇒ t(ground)=NULL;
C5          t(place_pos)≠t(place_neg);
C6          t(place_neg)≠t(ground);
C7          t(place_pos)≠t(ground)
      endtutp;
      keys {{channel,use_n,connecttime}}
    endtatp;
```

```
tatp electrode_imp =                      electrode impedance
  tutp T-elimp =                          (section 6.4)
    obcar F-elimp =
        channel      : nr;                channel used for measurement
        use_n        : nr;                usage number
        session_n    : nr;                session number
        imptime      : ltime;             long-integer time of impedance measurement
        placement    : chs(72);           placement of measured electrode
        ground       : chs(72);           placement of neutral electrode
        impedance    : pos ;              impedance of measured electrode
        unit         : imp;               unit
    endobcar;
    tuc t(placement)≠t(ground)
  endtutp;
  keys {{session_n,imptime,placement},{use_n,imptime,placement}}
endtatp;
```

```
tatp raw_fysdata =                                information on raw physiologic data
  tutp rawfy =                                    (section 6.5.1)
    obcar rawfy =
        raw_n          : nr;                      number of raw data
        dat_code       : chs(10);                 code for type of monitoring
        begintime      : ltime;                   long-integertime of start monitoring
        endtime        : ltime;                   long_integertime of end monitoring
        use_n          : nr;                      usage number of monitoring equipment
        sample_freq    : ℝ;                       sample frequency
        unit           : freq;                    unit of frequency
        session_n      : nr;                      session number
        file           : chs(72);                 reference to file with raw data
        format         : chs(72);                 format code of data
        medium         : chs(72);                 medium on which file is stored
    endobcar;
    tuc t(begintime) < t(endtime)
  endtutp;
  keys {{raw_n}}
endtatp;


tatp datatypes =                                  description of data types or curves
  tutp T-daty =                                   (section 6.5.1)
    obcar F-daty =
        dat_code       : Chs(10);                 code for curve
        description    : chs(72);                 description
    endobcar;
  endtutp;
  keys {{dat_code}}
endtatp;
```

```
tatp proc_data =                          signal values
  tutp proda =                            (section 6.5.2)
    obcar proda =
        proc_n       : nr;                number of processed data
        x_value      : ℝ;                 value of x-coordinate
        y_value      : ℝ;                 value of y-coordinate
    endobcar;
  endtutp;
  keys {{proc_n,x_value}}
endtatp;
```

```
tatp proc_infodata  =                    information on derived (processed) data
  tutp T-proin =                         (section 6.5.2)
    obcar F-proin =
        proc_n        : nr;              number of processed data set
        dat_code      : chs(10);         code for type of curve
        dataproc_n    : nr;              number of dataprocessing technique
        valid_code    : chs(10);         code for validity processed data set
        x_unitcode    : chs(10);         unit of x-coordinate
        y_unitcode    : chs(10);         unit of y-coordinate
        inproc_n      : nr;              number of originating data set
        begin_int     : ℝ;               begin of used interval
        end_int       : ℝ;               end of used interval
        begintime     : ltime;           long-integer begin time of used interval
        endtime       : ltime;           long-integer end time of used interval
        raw_n         : nr;              number of original raw data
        session_n     : nr;              session number
    endobcar;
    tuc t(begin_int) < t(end_int)
  endtutp;
```

C8 **tac** $\{\ t \ \mid t \in D \wedge t(\text{inproc\_n}) \neq \text{NULL}\ \} \infty \{(\text{proc\_n}\ ;\ \text{inproc\_n})\} \subseteq D \upharpoonright \{\text{proc\_n}\}$

keys {{proc_n}}

endtatp;

```
tatp dataprocessing =                              dataprocessing information
  tutp T-dapro =                                   (section 6.5.2)
    obcar F-dapro =
        dataproc_n    : nr;                        number of dataprocessing
        char_code     : chs(10);                   code for characterisation field
        value         : chs(72);                   value of characterisation field
    endobcar;
  endtutp;
  keys {{dataproc_n,char_code}}
endtatp;

tatp characteristics =                             characterisation fields
  tutp T-char =                                    (section 6.5.2)
    obcar F-char =
        char_code     : chs(10);                   code for characterisation field
        description   : chs(72);                   description
    endobcar;
  endtutp;
  keys {{char_code}}
endtatp;

tatp stud_procdata =                               study information on processed data
  tutp T-stpro =                                   (section 6.5.2)
    obcar F-stpro =
        study_code    : ['aa-001'..'zz-999'];      study code
        proc_n        : nr;                        number of processed data
        session_n     : nr;                        session number
    endobcar;
  endtutp;
  keys {{study_code,proc_n}}
endtatp;
```

```
dbtp emdabs =
 dbcar DK-em =
   subi              : subj_iddata;
   subd              : subj_demogrdata;
   subh              : subj_histdata;
   surg              : surgerydata;
   sess              : sessiondata;
   inst              : institutions;
   stud              : studydata;
   gran              : grants;
   stinf             : stud_infodata;
   stgra             : stud_grantdata;
   stins             : stud_insdata;
   pers              : personnel;
   func              : functions;
   sinf              : sess_infodata;
   spers             : sess_personneldata;
   sstud             : sess_studdata;
   pmed              : pre_medication;
   psess             : pre-sessdata;
   meast             : measure_type;
   drug              : drugdata;
   uni               : units;
   eve               : events;
   drev              : drugevents;
   flui              : fluids;
   druco             : drugconc;
   lab               : labresults;
   ande              : anest_depth;
   prot              : protocol;
   equ               : equipment;
```

```
    elepl             : electrode_place;
    equus             : equipment_use;
    stuse             : stud_usedata;
    mont              : montage;
    elec              : electrodes;
    adqua             : adjust_quant;
    sett              : settings;
    chan              : channels;
    elimp             : electrode_imp;
    proin             : proc_infodata;
    vali              : validity;
    proda             : proc_data;
    rawfy             : raw_fysdata;
    dapro             : dataprocessing;
    stpro             : stud_procdata;
    char              : characteristics;
    daty              : datatypes;
enddbcar;
dac X(subi) ↾ {sub_code} = X(subd) ↾ {sub_code};          subject-session dac's
    X(subd) ↾ {sub_code} = X(sess) ↾ {sub_code};
    X(subh) ↾ {sub_code} ⊆ X(subd) ↾ {sub_code};
    X(sess) ↾ {surgery_code} ⊆ X(surg) ↾ {surgery_code};
    X(sess) ↾ {instcode} = X(inst) ↾ {instcode};
    X(stins) ↾ {instcode} = X(inst) ↾ {instcode};
    X(stins) ↾ {study_code} = X(stud) ↾ {study_code};
    X(stinf) ↾ {study_code} ⊆ X(stud) ↾ {study_code};
    X(stgra) ↾ {study_code} = X(stud) ↾ {study_code};
    X(stgra) ↾ {grant_code} = X(gran) ↾ {grant_code};
    X(spers) ↾ {study_code} = X(stud) ↾ {study_code};
    X(spers) ↾ {func_code} = X(func) ↾ {func_code};
    X(spers) ↾ {pers_code} = X(pers) ↾ {pers_code};
```

{ t $\upharpoonright$ {study_code} $\mid$ t $\in$ X(stud) $\land$ t(study_code) $\neq$ 'a-000' }
= X(sstud) $\Vert\!\upharpoonright$ {study_code};
X(stud) $\Vert\!\upharpoonright$ {pers_code} $\subseteq$ X(pers) $\Vert\!\upharpoonright$ {pers_code};
{ t $\upharpoonright$ {session_n,study_code} $\mid$ t $\in$ X(spers) $\land$ t(study_code) $\neq$ 'a-000' }
= X(sstud) $\Vert\!\upharpoonright$ {session_n,study_code};
X(sinf) $\Vert\!\upharpoonright$ {session_n,study_code} $\subseteq$ X(sstud) $\Vert\!\upharpoonright$ {session_n,study_code};
X(pmed) $\Vert\!\upharpoonright$ {session_n} = X(sess) $\Vert\!\upharpoonright$ {session_n};
X(pmed) $\Vert\!\upharpoonright$ {drug_code} $\subseteq$ X(drug) $\Vert\!\upharpoonright$ {drug_code};
X(psess) $\Vert\!\upharpoonright$ {session_n} = X(sess) $\Vert\!\upharpoonright$ {session_n};
X(psess) $\Vert\!\upharpoonright$ {type_code} $\subseteq$ X(meast) $\Vert\!\upharpoonright$ {type_code};
X(pmed) $\Vert\!\upharpoonright$ {unit_code} $\subseteq$ X(uni) $\Vert\!\upharpoonright$ {unit_code};
X(psess) $\Vert\!\upharpoonright$ {unit_code} $\subseteq$ X(uni) $\Vert\!\upharpoonright$ {unit_code};
$\forall_{t \in X(\text{subd})} : \exists_{u \in X(\text{subi})}$: t(type) $\neq$ 'human' $\land$
u(sub_code)=t(sub_code) $\Rightarrow$ (u(ss_number)=NULL $\land$
u(first_name)=NULL $\land$ u(initials)=NULL);
$\forall_{t \in X(\text{subd})} : \forall_{u \in X(\text{subh})}$: t(sub_code)=u(sub_code) $\Rightarrow$
u(entry_date) $\geq$ t(date_birth);
$\forall_{t \in X(\text{subd})} : \forall_{u \in X(\text{subi})}$: t(sub_code)=u(sub_code) $\Rightarrow$
u(entry_date) $\geq$ t(date_birth); **end of subject-session dac's**

C9 X(sess) $\Vert\!\upharpoonright$ {session_n,begintime} $\subseteq$ (X(eve) $\Vert\!\upharpoonright$ {session_n,long_intime})
∞{(session_n;session_n),(begintime;long_intime)};

C10 X(sess) $\Vert\!\upharpoonright$ {session_n,endtime} $\subseteq$ (X(eve) $\Vert\!\upharpoonright$ {session_n,long_intime})
∞{(session_n;session_n),(endtime;long_intime)};

C11 X(eve) $\Vert\!\upharpoonright$ {session_n} = X(sess) $\Vert\!\upharpoonright$ {session_n};

C12 X(drev) $\Vert\!\upharpoonright$ {session_n,long_intime} $\subseteq$ X(eve) $\Vert\!\upharpoonright$ {session_n,long_intime};
X(drev) $\Vert\!\upharpoonright$ {drug_code} $\subseteq$ X(drug) $\Vert\!\upharpoonright$ {drug_code};
X(flui) $\Vert\!\upharpoonright$ {session_n,long_intime} $\subseteq$ X(eve) $\Vert\!\upharpoonright$ {session_n,long_intime};
X(druco) $\Vert\!\upharpoonright$ {session_n,long_intime} $\subseteq$ X(eve) $\Vert\!\upharpoonright$ {session_n,long_intime};
X(druco) $\Vert\!\upharpoonright$ {drug_code} $\subseteq$ X(drug) $\Vert\!\upharpoonright$ {drugcode};
X(lab) $\Vert\!\upharpoonright$ {session_n,long_intime} $\subseteq$ X(eve) $\Vert\!\upharpoonright$ {session_n,long_intime};
X(lab) $\Vert\!\upharpoonright$ {type_code} $\subseteq$ X(meast) $\Vert\!\upharpoonright$ {type_code};

X(lab) ‖↾ {unit_code} ⊆ X(uni) ‖↾ {unit_code};
X(ande) ‖↾ {session_n,long_intime} ⊆ X(eve) ‖↾ {session_n,long_intime};
X(ande) ‖↾ {protocol,estimation} ⊆ X(prot) ‖↾ {protocol,estimation};
X(ande) ‖↾ {valid_code} ⊆ X(vali) ‖↾ {protocol};
C13 X(equus) ‖↾ {equip_code) = X(equ) ‖↾ {equip_code};
X(equus) ‖↾ {instcode) ⊆ X(inst) ‖↾ {instcode};
C14 X(equus) ‖↾ {session_n} = X(sess) ‖↾ {session_n};
X(stuse) ‖↾ {use_n} = X(equus) ‖↾ {use_n};
X(stuse) ‖↾ {session_n} = X(sess){session_n};
C15 { t ↾ {session_n,study_code} | t ∈ X(stuse) ∧ t(study_code)≠'aa-000'} =
X(sstud) ‖↾ {session_n,study_code};
X(mont) ‖↾ {placement} ⊆ X(elepl) ‖↾ {placement};
C16 X(mont) ‖↾ {session_n} = X(sess) ‖↾ {session_n};
X(mont) ‖↾ {elec_code} ⊆ X(elec) ‖↾ {elec_code};
X(adqua) ‖↾ {equip_code} ⊆ X(equ) ‖↾ {equip_code};
X(sett) ‖↾ {use_n} ⊆ X(equus) ‖↾ {use_n};
X(sett) ‖↾ {install_code} ⊆ X(infi) ‖↾ {install_code};
X(chan) ‖↾ {use_n} ⊆ X(equus) ‖↾ {use_n};
X(chan) ‖↾ {place_pos} ⊆ (X(elepl) ‖↾ {placement})∞{(place_pos;placement)};
X(chan) ‖↾ {place_neg} ⊆ (X(elepl) ‖↾ {placement})∞{(place_neg;placement)};
{ t ↾ {ground} | t ∈ X(chan) ∧ t(ground)≠NULL} ⊆ (X(elepl) ‖↾ {placement})∞
{(ground;placement)};
X(elimp) ‖↾ {use_n} ⊆ X(equus) ‖↾ {use_n};
X(elimp) ‖↾ {placement,session_n} ⊆ X(mont) ‖↾ {placement,session_n};
X(elimp) ‖↾ {ground,session_n} ⊆ (X(mont) ‖↾ {placement,session_n})∞
{(ground;placement),(session_n,session_n)};
X(proin) ‖↾ {x_unitcode} ⊆ (X(uni) ‖↾ {unit_code})∞{(x_unitcode;unit_code)};
X(proin) ‖↾ {y_unitcode} ⊆ (X(uni) ‖↾ {unit_code})∞{(y_unitcode;unit_code)};
X(proin) ‖↾ {raw_n} ⊆ X(rawfy) ‖↾ {raw_n};
X(proin) ‖↾ {dat_code} ⊆ X(daty) ‖↾ {dat_code};
X(proin) ‖↾ {session_n} ⊆ X(sess) ‖↾ {session_n};

```
        X(proin) |↾ {dataproc_n} = X(dapro) |↾ {dataproc_n};
        X(proin) |↾ {valid_code} ⊆ X(vali) |↾ {valid_code};
        X(proda) |↾ {proc_n} = X(proin) |↾ {proc_n};
        X(rawfy) |↾ {use_n} ⊆ X(equus) |↾ {use_n};
        X(rawfy) |↾ {session_n} = X(sess) |↾ {session_n};
        X(rawfy) |↾ {dat_code} ⊆ X(daty) |↾ {dat_code};
        X(dapro) |↾ {char_code} ⊆ X(char) |↾ {char_code};
        X(stpro) |↾ {proc_n} = X(proin) |↾ {proc_n};
        X(stpro) |↾ {session_n} ⊆ X(sess) |↾ {session_n};
        X(stpro) |↾ {study_code,session_n} ⊆ X(sstud) |↾ {study_code,session_n};
    enddbtp;
endtype.
```

REFERENCES

Brock, E.O. de (1989). De grondslagen van semantische databases (in Dutch). Schoonhoven: Academic Service.

Brock, E.O. de (1985). Database systemen 1 (deel 1) (in Dutch). Syllabus 1985. Eindhoven : Eindhoven University of Technology, 1986. Dictaatnr. 2405.

Brock, E.O. de (1985). Database systemen 1 (deel 2) (in Dutch). Syllabus 1985. Eindhoven : Eindhoven University of Technology, 1986. Dictaatnr. 2407.

Brower, R.W. , Katen, H.J. ten , Meester, G.T. (1984). Problems and pitfalls in a clinical research data management system. Computer Programs in Biomedicine, Vol 19 (1984), p. 13-30.

Budd, J.R. , Warwick, W.J. , Wielenski, C.L. , Finkelstein, S.M. (1988). A medical Information Relational Database System (MIRDS). Computers and Biomedical Research, vol 21 (1988), p. 419-433.

Cluitmans, P.J.M. (1990). Neurophysiological Monitoring of Anesthetic Depth. Eindhoven: Eindhoven University of Technology. Ph.D. thesis.

Date, C.J. (1986). An introduction to Database Systems. Vol. I, 4th ed. Reading, Mass,: Addison-Wesley, 1986.

Herwijnen, G.J. van (1988). Het Converteren van een Klinische Database van dBase III naar Oracle (in Dutch). Eindhoven: Fac. Electrical Engineering, Eindhoven University of Technology, 1988. M.Sc. thesis.

Hilderink, H.G.M. (1990). Datamodellen in de gezondheidszorg (in Dutch). Verslag studiedag, Bunnik, 20 sept. 1990. onder redactie van H.G.M. Hilderink. VMBI Datamodelwerkgroep, p/a RZI, Plotterweg 14, Amersfoort.

Jong, P.G.M. de (1986). Ontwikkeling van een event recording module voor een event recording en data acquisition systeem (in Dutch). Eindhoven: Fac. Electrcal Engineering, Eindhoven University of Technology, 1986. M.Sc. thesis.

Korlaar, A. (1989). Databasesystemen voor Bdk (in Dutch). Syllabus 1989. Eindhoven: Eindhoven University of Technology, 1989. Dictaatnr. 2424.

Lutz, M.W. (1986). The design and evaluation of a methodology to generate programs which interface with a medical database (automatic programming, query sytems). Duke University, 1986. Ph.D. thesis. University Microfilms Publication No. AAC8700024.

Perry, J.T. , Lateer, J.G. (1989). Werken met ORACLE (translated from Englisch to Dutch). Arnhem: Sybex, 1989.

Remmen, F. (1982). Databases: grondslagen voor de logische structuur (in Dutch). Den Haag: Academic Service, 1982.

Theisen, G.J. , Cluitmans, P.J.M. , Beneken, J.E.W. , Conlon, M. , Grundy, B.L. (1986). EMDABS: A Multi-institutional Research Database System for Electrophysiologic Monitoring of the Nervous System. In: MEDINFO 86: Proc. 5th Conf. on Medical Informatics, Washington, 26-30 Oct. 1986.Ed. by R. Salamon , B. Blum and M. Jørgensen. Amsterdam: North-Holland, 1986. IFI P World Conference Series on Medical Informatics, vol.5. P.565-567 (part 1).

Testa, M.A. , Simonson, D.C. (1985). The design and structure of clinical research information systems: Implications for data retrieval and statistical analyses. In: Proc. 18th Hawaii Int. Conf. on System Sciences, Honolulu, 2-4 Jan. 1985. vol.3: Medical Information Processing. Ed. by T.M. Walker. North Hollywood, Cal.: Western Periodicals, 1985. P.327-336. Processing, p. 327-336.

Yao, H.H. (1989). Extended Relational Operators for Statistical Data Manipulations in Medical Databases. Computers and Biomedical Research, vol. 22 (1989), p. 516-531.

ISSN 0167-9708
Coden: TEUEDE

(222) Jóźwiak, L.
THE FULL-DECOMPOSITION OF SEQUENTIAL MACHINES WITH THE SEPARATE REALIZATION OF THE NEXT-STATE AND OUTPUT FUNCTIONS.
EUT Report 89-E-222. 1989. ISBN 90-6144-222-2

(223) Jóźwiak, L.
THE BIT FULL-DECOMPOSITION OF SEQUENTIAL MACHINES.
EUT Report 89-E-223. 1989. ISBN 90-6144-223-0

(224) Book of abstracts of the first Benelux-Japan Workshop on Information and Communication Theory, Eindhoven, The Netherlands, 3-5 September 1989.
Ed. by Han Vinck.
EUT Report 89-E-224. 1989. ISBN 90-6144-224-9

(225) Hoeijmakers, M.J.
A POSSIBILITY TO INCORPORATE SATURATION IN THE SIMPLE, GLOBAL MODEL OF A SYNCHRONOUS MACHINE WITH RECTIFIER.
EUT Report 89-E-225. 1989. ISBN 90-6144-225-7

(226) Dahiya, R.P. and E.M. van Veldhuizen, W.R. Rutgers, L.H.Th. Rietjens
EXPERIMENTS ON INITIAL BEHAVIOUR OF CORONA GENERATED WITH ELECTRICAL PULSES SUPERIMPOSED ON DC BIAS.
EUT Report 89-E-226. 1989. ISBN 90-6144-226-5

(227) Bastings, R.H.A.
TOWARD THE DEVELOPMENT OF AN INTELLIGENT ALARM SYSTEM IN ANESTHESIA.
EUT Report 89-E-227. 1989. ISBN 90-6144-227-3

(228) Hekker, J.J.
COMPUTER ANIMATED GRAPHICS AS A TEACHING TOOL FOR THE ANESTHESIA MACHINE SIMULATOR.
EUT Report 89-E-228. 1989. ISBN 90-6144-228-1

(229) Oostrom, J.H.M. van
INTELLIGENT ALARMS IN ANESTHESIA: An implementation.
EUT Report 89-E-229. 1989. ISBN 90-6144-229-X

(230) Winter, M.R.M.
DESIGN OF A UNIVERSAL PROTOCOL SUBSYSTEM ARCHITECTURE: Specification of functions and services.
EUT Report 89-E-230. 1989. ISBN 90-6144-230-3

(231) Schemmann, M.F.C. and H.C. Heyker, J.J.M. Kwaspen, Th.G. van de Roer
MOUNTING AND DC TO 18 GHz CHARACTERISATION OF DOUBLE BARRIER RESONANT TUNNELING DEVICES.
EUT Report 89-E-231. 1989. ISBN 90-6144-231-1

(232) Sarma, A.D. and M.H.A.J. Herben
DATA ACQUISITION AND SIGNAL PROCESSING/ANALYSIS OF SCINTILLATION EVENTS FOR THE OLYMPUS PROPAGATION EXPERIMENT.
EUT Report 89-E-232. 1989. ISBN 90-6144-232-X

(233) Nederstigt, J.A.
DESIGN AND IMPLEMENTATION OF A SECOND PROTOTYPE OF THE INTELLIGENT ALARM SYSTEM IN ANESTHESIA.
EUT Report 90-E-233. 1990. ISBN 90-6144-233-8

(234) Philippens, E.H.J.
DESIGNING DEBUGGING TOOLS FOR SIMPLEXYS EXPERT SYSTEMS.
EUT Report 90-E-234. 1990. ISBN 90-6144-234-6

(235) Heffels, J.J.M.
A PATIENT SIMULATOR FOR ANESTHESIA TRAINING: A mechanical lung model and a physiological software model.
EUT Report 90-E-235. 1990. ISBN 90-6144-235-4

(236) Lammers, J.O.
KNOWLEDGE BASED ADAPTIVE BLOOD PRESSURE CONTROL: A Simplexys expert system application.
EUT Report 90-E-236. 1990. ISBN 90-6144-236-2

(237) Ren Qingchang
PREDICTION ERROR METHOD FOR IDENTIFICATION OF A HEAT EXCHANGER.
EUT Report 90-E-237. 1990. ISBN 90-6144-237-0

Eindhoven University of Technology Research Reports
Faculty of Electrical Engineering

ISSN 0167-9708
Coden: TEUEDE

(238) Lammers, J.O.
THE USE OF PETRI NET THEORY FOR SIMPLEXYS EXPERT SYSTEMS PROTOCOL CHECKING.
EUT Report 90-E-238. 1990. ISBN 90-6144-238-9

(239) Wang, X.
PRELIMINARY INVESTIGATIONS ON TACTILE PERCEPTION OF GRAPHICAL PATTERNS.
EUT Report 90-E-239. 1990. ISBN 90-6144-239-7

(240) Lutgens, J.M.A.
KNOWLEDGE BASE CORRECTNESS CHECKING FOR SIMPLEXYS EXPERT SYSTEMS.
EUT Eeport 90-E-240. 1990. ISBN 90-6144-240-0

(241) Brinker, A.C. den
A MEMBRANE MODEL FOR SPATIOTEMPORAL COUPLING.
EUT Report 90-E-241. 1990. ISBN 90-6144-241-9

(242) Kwaspen, J.J.M. and H.C. Heyker, J.I.M. Demarteau, Th.G. van de Roer
MICROWAVE NOISE MEASUREMENTS ON DOUBLE BARRIER RESONANT TUNNELING DIODES.
EUT Report 90-E-242. 1990. ISBN 90-6144-242-7

(243) Massee, P. and H.A.L.M. de Graaf, W.J.M. Balemans, H.G. Knoopers, H.H.J. ten Kate
PREDESIGN OF AN EXPERIMENTAL (5-10 MWt) DISK MHD FACILITY AND PROSPECTS OF COMMERCIAL (1000 MWt) MHD/STEAM SYSTEMS.
EUT Report 90-E-243. 1990. ISBN 90-6144-243-5

(244) Klompstra, Martin and Ton van den Boom, Ad Damen
A COMPARISON OF CLASSICAL AND MODERN CONTROLLER DESIGN: A case study.
EUT Report 90-E-244. 1990. ISBN 90-6144-244-3

(245) Berg, P.H.G. van de
ON THE ACCURACY OF RADIOWAVE PROPAGATION MEASUREMENTS: Olympus propagation experiment.
EUT Report 90-E-245. 1990. ISBN 90-6144-245-1

(246) Maagt, P.J.L. de
A SYNTHESIS METHOD FOR COMBINED OPTIMIZATION OF MULTIPLE ANTENNA PARAMETERS AND ANTENNA PATTERN STRUCTURE.
EUT Report 90-E-246. 1990. ISBN 90-6144-246-X

(247) Jóźwiak, L. and T. Spassova-Kwaaitaal
DECOMPOSITIONAL STATE ASSIGNMENT WITH REUSE OF STANDARD DESIGNS: Using counters as sub-machines and using the method of maximal adjacensies to select the state chains and the state codes.
EUT Report 90-E-247. 1990. ISBN 90-6144-247-8

(248) Hoeijmakers, M.J. and J.M. Vleeshouwers
DERIVATION AND VERIFICATION OF A MODEL OF THE SYNCHRONOUS MACHINE WITH RECTIFIER WITH TWO DAMPER WINDINGS ON THE DIRECT AXIS.
EUT Report 90-E-248. 1990. ISBN 90-6144-248-6

(249) Zhu, Y.C. and A.C.P.M. Backx, P. Eykhoff
MULTIVARIABLE PROCESS IDENTIFICATION FOR ROBUST CONTROL.
EUT Report 91-E-249. 1991. ISBN 90-6144-249-4

(250) Pfaffenhöfer, F.M. and P.J.M. Cluitmans, H.M. Kuipers
EMDABS: Design and formal specification of a datamodel for a clinical research database system.
EUT Report 91-E-250. 1991. ISBN 90-6144-250-8