# LOGICAL FOUNDATIONS OF CONCEPTUAL AND DATABASE MODELLING

*Marie Duží*

*VSB-Technical University Ostrava*

**http://www.cs.vsb.cz/duzi/**

# *Content*

# 1. Introduction

The development of information systems (IS) in the last two decades has been often characterised as the „information backlog“. The users keep complaining: „We never get what we want. And if we eventually get it, it is always too late“. And the designers of IS answer: „They (the users) never know what they want. They keep changing their demands and they never express them exactly“. As a way out from this crisis many CASE products have been offered and available in the market, declaring the ability of „miraculously“ fast creation of an indubitably „correct“ information system. As a matter of fact, these products make it possible to rapidly create a *prototype* of the system, which in itself does not solve the problem. CASE products are good *tools* for the rapid design and coding of the IS, but as an every tool they can be useful only when we know *how and why* to use them. Actually, theoretical approaches to conceptual modelling are not applied in the CASE tools available, and the work of practitioners is mostly driven by intuition. However, using CASE products properly and building a „correct“ IS require a deep understanding of data semantics, i.e. a thorough *conceptual analysis* of IS.

The creation of IS can be characterised as passing through the following phases (which is often called the ‘life cycle’ of the IS):

- Conceptual analysis (conceptual level)
  In this phase we aim at conceptual understanding and rendering the modelled part of reality (the Universe of Discourse; UoD). We try to find out which information about which objects of interest the IS should keep and provide, and in which way should the IS manipulate the data. The result of this phase is the *conceptual schema (model)* of the organisation together with the specification of the functions that the system should perform. Conceptual schema should be totally independent of the intended DBMS (database management system) and intelligible for users.

- The design of IS (logical level)
  In this phase we design a (logical) database schema of the system that should still be independent of the concrete DBMS in which it will be implemented. However, the logical database schema is partly determined by a data model (nowadays actually exclusively a relational one) that the desired DBMS is supposed to be a kind of. The specification of the functions is made more detailed with respect to the designed database schema, but we still concentrate on the problems of *what* the given program shall perform without saying *how* will it be performed.

- Implementation and debugging (physical level)
  The designed database schema is implemented in the chosen DBMS and the specified programs are coded. After debugging the resulting product is ready to be installed in the organisation.

- Usage and maintenance
  This is the concluding phase of the system life cycle. If the previous phases were not performed correctly and their results were not thoroughly documented, fatal problems appear in this phase that may depreciate the efforts of the whole work.

The phase of the conceptual analysis of the system has been traditionally neglected or underestimated, though it is just this stage that is crucial for the development of the whole system. The majority of errors have their origin in wrong conceptual decisions and they are just these errors the correction of which is very difficult not to say impossible. Another

consequence of underestimating a thorough conceptual analysis is the fact that the resulting system is „stiff“, not adaptable to permanently changing user requirements.

Yet in the last few years a shift of research interests is observable: from the „classical“ record-oriented database models, the typical representative of which is the relational data model [Codd 1970] with its rich mathematical-logical theory, to the semantic data models [Chen 1976], [Abiteboul 1987], [Hull 1987], [Hammer 1981], [Becker 1998a, b, c] and object-oriented data models [Alagic 1999], [Cattell 1997], [Halpin 1998], and we can even say that generally a formal work on conceptual modelling is gradually becoming a hot topic nowadays. This work can be characterised as a contribution to this topic. To summarise our goals we now quote from [Halpin 1998] (emphasises MD):

**Conceptual modelling language criteria**

A modelling method comprises a language and also a procedure for using the language to construct models. Written languages may be graphical (diagrams) and/or textual. Conceptual models portray applications at a fundamental level, using terms and concepts *familiar to the application users*. In contrast, logical and physical models specify underlying database structures to be used for implementation, and external models specify user interaction details (e.g. design of screen forms and reports). The following criteria provide a useful basis for evaluating conceptual modelling methods.

- Expressibility
- Clarity
- Semantic stability
- Semantic relevance
- Validation mechanisms
- Abstraction mechanisms
- Formal foundation

*The expressibility* of a language is a measure of what it can be used to say. Ideally, a conceptual language should be able to model all conceptually relevant details about the application domain. This is called the 100% Principle [ISO 1982]. HIT (see Chapters 3 – 5) and ORM (see Chapter 2) are primarily methods for modelling and querying an information system at the conceptual level, and for mapping between conceptual and logical levels. The focus is on data modelling, since the data perspective is more stable and it provides a formal foundation on which operations can be defined.

*The clarity* of a language is a measure of how easy it is to understand and use. To begin with, the language should be unambiguous. Ideally, the meaning of diagrams or textual expressions in the language should be *intuitively obvious*. The language notations should be easily learnt and remembered.

*Semantic stability* is a measure of how well models or queries expressed in the language retain their original intent in the face of changes to the application.

*Semantic relevance* requires that only conceptually relevant details need to be modelled. Any aspect irrelevant to the meaning (e.g. implementation choices, machine efficiency) should be avoided. This is called the *Conceptualisation Principle* [ISO 1982].

*Validation mechanisms* are ways in which domain experts can check whether the model matches the application. For example, static features may be checked by verbalisation and multiple instantiation, and dynamic features may be checked by simulation.

*Abstraction mechanisms* are ways in which unwanted details may be removed from immediate consideration. This is especially important with large models.

*A formal foundation* ensures that models are unambiguous and executable. One particular benefit is to allow *formal proofs of equivalence and implication* between alternative models for the same application.

The core of this work can be characterised as describing the HIT data model which, in our opinion, meets all the above criteria.

A possible objection to the presented work could be formulated as follows: It is not quite clear how to classify it within the system of mathematical/logical disciplines. It does not concern pure (philosophical) logic nor the theory of databases as commonly conceived. From this point of view it is rather an interdisciplinary work. I would like to show that this interdisciplinary character, though being apparently a negative feature, actually corresponds well with current trends in logic as well as in the database theory. Moreover, applying these new modern trends from both disciplines brings some promising assets.

The study is organised as follows: First, after a brief survey of current semantic (conceptual) data models in Section 2, Section 3 provides a critical description, comparison and in a way integration of two approaches to the problem of conceptual modelling: the 'Finnish school' represented by the COMIC system [Kangassalo 1993] for conceptual modelling and the 'Czech school' represented by the HIT data model [Zlatuška 1986], [Duží 1986, 1992, 1997, 1999]. Both the approaches have much in common. They can be characterised as a contribution to the theory of conceptual modelling. They both stress the importance of *users involvement* in the conceptual analysis of the system based on the careful analysis of natural language expressions describing the objects of interest. They both concentrate on the relatively stable and in a way basic part of the system, namely the concepts of the objects the data of which will be recorded (data analysis), leaving the description of the permanently changing „behaviour" of the system (functional analysis) as a secondary problem prior to which is the conceptual description of data structures. Afterwards, Section 4 deals with the problem of semantic information connected with data. This section is partly taken over the [Duží 1991]. Nevertheless, many ideas have since 1991 'gurgled out' and some results and proofs have been corrected. In our opinion a solid theoretical background is still a lacking point of the majority of conceptual data models, and since the HIT data model is the (only ?) one that does not suffer from this theoretical / logical disregard, we did not hesitate to partly repeat this part of [Duží 1991], not only for the completeness. We introduce and use rigorous theoretical tools for studying the semantics of general data structures in the context of conceptual modelling. Using just one modelling construct — HIT attribute — that via generalisation of usual functional constructs covers all the 'classical' constructs, makes it possible to use functional approach to data modelling based on a unique formal apparatus, namely the 'language of TIL (Transparent Intensional Logic) constructions' throughout the whole study. We develop a solid logical foundation and integrating base upon which a theoretical investigation of relative informational capability of general schemata can be based. Last but not least, we define a minimum data structure informationally equivalent with a general database conceptual system that can serve as a stable 'kernel' of the whole system. Finally, in Section 5, the methodology of transforming the HIT conceptual schema via the Chen's like E-R schema [Chen 1976] into the relational schema [Codd 1970] in the 4$^{th}$ normal form is provided, thus giving a practical means for the database design and implementation.

# 2. Semantic data models

As stated in Chapter 1, we need to be able to first design a conceptual schema, one that accurately and completely defines business rules in a way that our users can understand. This conceptual layer is free of implementation details such as database vendor and schema implementation (Relational vs. Object-Relational vs. Object-Oriented, etc.). We can then express that conceptual knowledge in a somewhat implementation-biased and less abstract logical notation. Finally, we express the model in a physical notation. In this fashion, the logical and physical schemas are nothing more than an abstraction of the *conceptual schema*, which contains all of the information we need to accurately express the business rules and data requirements. We also need to communicate better with the users. For example, say you are validating your model with your users using terms like entities, attributes, and relationships, or even worse, foreign keys, referential integrity, and tuples, it may happen that your users are vigorously nodding while giving you a blank look. Odds are, they don't have a complete understanding of what you are talking about. Most of the time, they won't even ask for clarification. The reason for this is simple: we often use a very obscure language that most people don't understand (or even want to understand).

What follows is a brief overview of semantic, or better conceptual data models that less or more overcome the above described difficulties. We do not intend to give an exhausting list of them; there are so many models which are called semantic, that such a task would be completely out of the scope of this work. We just want to illustrate the typical principles, constructs and building blocks of such models, so as to be able to make a comparison and summary.

## *2.1. Entity-Relationship model*

The *Entity-Relationship* (E-R) model, developed by P. Chen [Chen 1976], is perhaps the best known of the conceptual and view approaches that has become a standard nowadays, and nearly all the available CASE products are based on the E-R methodology with its graphical support. Though it is a familiar ground to almost everybody dealing with the problems of database modelling, let us very briefly recapitulate the principles of this model. It employs only three basic modelling constructs, namely *entities, relationships and attributes*. These are (in E-R) described as follows:

- An *entity* is a „thing" of interest in a database and something that can be clearly defined. In other words, it is an object of the real world that can independently exist and that is uniquely distinguishable from the others. In the case of an university, examples of entities might be a 'student', a 'professor', a 'course', etc.
- A *relationship* is an association among two or more entities. For example, 'a student takes courses', 'a professor lectures courses' are associations between student and courses, professor and courses, respectively.
- *Attribute* is a function assigning to an entity or to a relationship a value that expresses some important property or characteristics that can be identified for both an entity and a relationship. For example, an id-number of a student could be an attribute of the entity 'student', and the obtained grade an attribute of the relationship 'student has enrolled in a course'.

These constructs are described in a rather intuitive way, as it is, after all, in almost all the current data models. Precise explication and definition will be provided in Section 3.2. Semantics are captured by the E-R model in several ways [Storey 1988]. First, each entity plays a certain role in a relationship. For instance, a student has enrolled in a course, a course is taken by students. Semantics are captured another way through the concepts of an entity set (sometimes type) and a relationship set (type). Members of an entity set or a relationship set satisfy certain conditions characteristic of that set. In other words, each member of a set has certain property by means of which the set is defined. There are, however, situations where the E-R model fails to capture semantics. The model does not allow for the representation of relationships between two relationships or between an entity and a relationship. When using the E-R model, clear identification and distinction of the main constructs is required at the very beginning of the design process, which may be sometimes a stumbling block of the analysis. Especially determining relationship sets may become difficult; for instance, is a loan an entity or a relationship between a person and a thing that has been loaned?

Each entity has to be uniquely identifiable; to this end an attribute (or a group of attributes) has to be assigned to the entity, namely its *identification key.* There may be several candidates for the role of an identification key. For instance, an employee can be identified by:

- first name, surname and the date of birth
- employee number (in a given organisation)
- identity card number
- social security number

Relationships can be of different types, which is recorded by *cardinality* ratio (or relationship degree), namely 1:1, 1:N and M:N. Our examples are of the type M:N (a student can be enrolled in more than one courses, a course is taken by more than one student). The cardinality of a relationship is sometimes expressed by statements like 'an entity of one type *uniquely determines* an entity of another type' [Howe 1989] in case of the N:1 cardinality (or that the former entity is a '*determiner'* for the latter). The terminology, however, is not unique. Some authors use the term '*functional dependency*', e.g. [Elmasri 1989].

There are some variants of this basic proposal, increasing the modelling power of the model; there may be some participation constraints specified for entities being members of relationships. They are *partial participation* and *total participation*. The former expresses the fact that the membership of an entity in the relationship is optional, the latter expresses an obligatory (mandatory) membership. For instance, the reality in an organisation may be such that a department can exist without employees, but an employee has to be taken on a department, he cannot be independent. In such a case it is often said that employees are *existentially dependent* on departments.

Besides 'normal' entity types, as they have been described above, there may also be such entity types that are not identified only by their own attributes but also by attributes of some other entity. In that case we speak about a *weak entity type* in contradistinction to the *strong (regular) entity type*. An instance of a weak entity type is identified by being in an obligatory relation to the instance of a strong entity type which is called its *owner*. In other words, a weak entity type has always an obligatory membership in the (identifying) relation to the strong entity type (its owner), hence it is existentially dependent on its owner. Imagine, for instance, an information system in which the data of several companies are recorded and each company has its own numbering of employees. In that case employee number is not a unique identification of an employee; it has to be identified also by the company. Employee

becomes a weak entity and it has an obligatory membership in the relationship 'employee is employed in a company'.

In some modifications of the model there are possibilities to record the so-called ISA hierarchies (or subtypes) of entity types. These relations will be precisely explicated and defined in section 3.2.5. Since in E-R literature they are defined only intuitively, let us be content for the moment with an informal description. Consider an entity type PERSON with attributes identity card number, name, date of birth, address, and so on. But if a person is a teacher, it is reasonable (and only in case of a teacher) to consider some other attributes like academic degree, or it is reasonable to bind a teacher in relationships like 'a teacher is lecturing a subject', 'subjects that a teacher can lecture', 'a teacher is a tutor of a student', etc. Students can also have their own special attributes like an average grade, number of credits, etc. Hence we introduce new entity types STUDENT and TEACHER into the schema, and claim that these types are subtypes of the type PERSON. Such a relation is called the *ISA relation* (from 'is a': each professor is a person, each student is a person) and it is reflexive, transitive and anti-symmetric. We can also speak about ISA hierarchy of entities ordered by this relation. Obviously, entities that are 'lower' in the hierarchy inherit attributes of the 'higher' entities.

Attributes of the basic model proposal are considered to be only atomic, i.e. attributes assigning to each entity (relationship) at most one (indivisible) value. Later versions allow to use more complex structured attributes composed of atomic ones. They are either *aggregated* attributes or *multivalued* attributes. A typical example of an aggregated attribute is the address of ... which consists of the state, town, ZIP code, street and street number. Such attributes may generally form a hierarchical structure (typical for COBOL records). Multivalued attributes assign a *set* of values to an entity (relationship). For instance 'names of authors of a title' is a multivalued attribute. Both the types of structured attributes may be combined. For instance, in a library IS we might have a structure (recorded in a linear way (used in many E-R models)): TITLE(...,AUTHORS(FIRST-NAME, SURNAME, NATIONALITY):Multi). However, such a modelling structure would not be probably designed in the best way, for it breaks the principles of the 3$^{rd}$ (4$^{th}$) normal form [Ullman 1988], see also Section 5 below. The data on authors are hidden under the titles, which may cause difficulties when manipulating authors (accessing via authors, updating, etc.). Nevertheless, such structures may be in some cases very useful and efficient.

## *2.2. IFO model*

The IFO model is a formal semantic database model that has been proposed by S. Abiteboul and R. Hull as a formally defined database model that combines fundamental principles of semantic database modelling in a coherent fashion.[1] Indeed, all the classical modelling constructs can be found here, though named in a rather different way. We are going to show the correspondence between IFO constructs and classical constructs. Moreover, IFO gives a mathematically formal definition of update propagation, and it is shown here that (under certain conditions) a correct update always exists.

The modelling building blocks of an IFO schema are called (object) *types* and *fragments*. Types model various objects structures of the application domain. Fragments represent functional relationships between types. *ISA relationships* are defined in fragments by either generalisation or specialisation. *IFO schemas* are directed graphs that are built by

[1] See [Abiteboul 1987]

combining fragments. Now we briefly describe these particular building blocks, and afterwards mention the update semantics in the IFO model.

### 2.2.1 Object types

The basis of any IFO schema is the representation of the various object structures, called *types.* There are three kinds of *atomic types* and two constructs for recursively building *derived types.*

One of the three atomic types is called *printable*, and it basically corresponds to a value type of the E-R model or a descriptive sort of the HIT model. In an IFO schema this type is indicated using a square node with the name of the type in it. The types that can be considered depend, of course, on the application (e.g. STRING, INTEGER, REAL, BOOLEAN, ...). The node is also provided with a name or label, like NAME, NUMBER, ...

The second atomic type is called *abstract*, and corresponds to an entity type or entity sort of the E-R model, HIT model, respectively. Informally it can be described as a set of objects in the real world that have no underlying structure (at least from the point of view of a designer or user). For instance the type PERSON is typically viewed as having no underlying structure though it may have many attributes and subtypes. This kind of object structure is represented in the schema by a diamond with an associated name. A *domain* of this type is a finite set of abstract symbols corresponding to particular *instances*, i.e. objects of the real world (people in our example).

The third (and last) atomic type is called *free,* and corresponds to the set of entities obtained via ISA relationship. Free atomic types can gain their actual type through the ISA relationship from other types. For instance the type STUDENT will inherit its structure from the type PERSON. Free types are represented in the IFO schema by using an empty circle.

The first of the two mechanism of building nonatomic derived types consists in forming finite *sets of objects* of a given structure. It corresponds to *collection* in the Hull, Yap's Format model [Hull 1984] or to *grouping clauses* in the Hammer, McLeod's SDM model [Hammer 1981]. In the HIT data model set construct is modelled by a type of characteristic function in the range of multivalued attributes. For instance a finite set of students enrolled in a course (called CLASS) is an 'object' having that particular structure. In the schema, this type is represented by a 'star vertex' (a circle provided with a star inside).

The other mechanism for constructing new types out of existing ones is *Cartesian product*. This has been called the *tuple* construct in the HIT model or *aggregation, composition* in the Format model and others. For instance a MOTOR-BOAT is an ordered pair of a HULL and a MOTOR. This type is represented by a 'cross-vertex' (a circle provided with a cross inside).

### 2.2.2. Fragments

The second main structural component of the IFO model is *fragment* which is the direct representation of functional relationships. It basically corresponds to HIT (unary) attributes. This representation of functions is closely related to the representation of functions in the Functional Data Model [Shipman 1981], with one key difference: In the IFO model a distinction is made between vertices serving the role of domain and vertices serving the role of range. This permits nested functions to be modelled in the IFO schema and a modular view of the IFO schema. Some integrity constraints on functions can be attached to the edge connecting two vertices of a fragment, namely total and singular. An example of a fragment is 'a class (a set of students) is enrolled in a course'.

### 2.2.3. ISA Relationships

The final structural component of the IFO model is the representation of *ISA relationships.* Intuitively, an ISA relationship from a type SUB to a type SUPER indicates that each object associated with SUB is associated with the type SUPER. This immediately implies that each function defined on the type SUPER is automatically defined on SUB, that is, functions of SUPER are *inherited* by SUB. Similarly as in other semantic models, ISA relationships are acquired by *specialisation* and *generalisation.* Specialisation can be used to define possible roles of members of a given type (e.g. a person might be a student, a person might be an employee). Specialisation can be overlapping: a person might be both a student and an employee. In contrast, using generalisation, distinct, pre-existing types are combined to form a new type. It is typical to require that a generalised supertype be covered by its subtypes that are disjoint. An IFO graph is a directed acyclic graph, where types are vertices and ISA relations are represented by arrow edges; arrow-head points at a supertype and arrow-tail at a subtype. Specialisation is marked by an 'empty arrow' and generalisation by a 'full arrow'. In specialisation, the type of the vertex is *inherited* 'top-down', from supertype to subtype. To prevent a type conflict, the tail (SUB) must be of free type. On the other hand, in generalisation, the type of the vertex is inherited 'bottom-up'. Hence the head must be of free type.

### 2.2.4. IFO Schemas

An *IFO graph* is defined as a directed graph S = (V, E) such that

(1) E is the disjoint union of three sets $E_O$ (object edges), $E_G$ (generalisation edges) and $E_S$ (specialisation edges)

(2) (V, $E_O$) is a 'forest' of fragments called the *fragment* of S. The roots of the fragments of S are called *primary vertices.*

Note that the primary vertices of a schema S will identify the types of the entity sets of prime interest that are represented by S, and also the top-level functions defined on these types. There are some conditions imposed on the specialisation and generalisation edges. To express them a concept of a 'reversal' graph must be defined first:

The *'reversal' graph* of the graph S is the graph $(V, \rho(E_O) \cup \rho(E_G) \cup \rho(E_S))$, where $\rho(E)$ is the set $\{(q,p) \mid (p,q) \in E\}$. There are five ISA rules defined on this graph:

ISA1: Each free vertex is either the tail of at least one specialisation edge, or the head of at least one generalisation edge, but not both.

ISA2: For each specialisation edge, the tail is free and the head is primary.

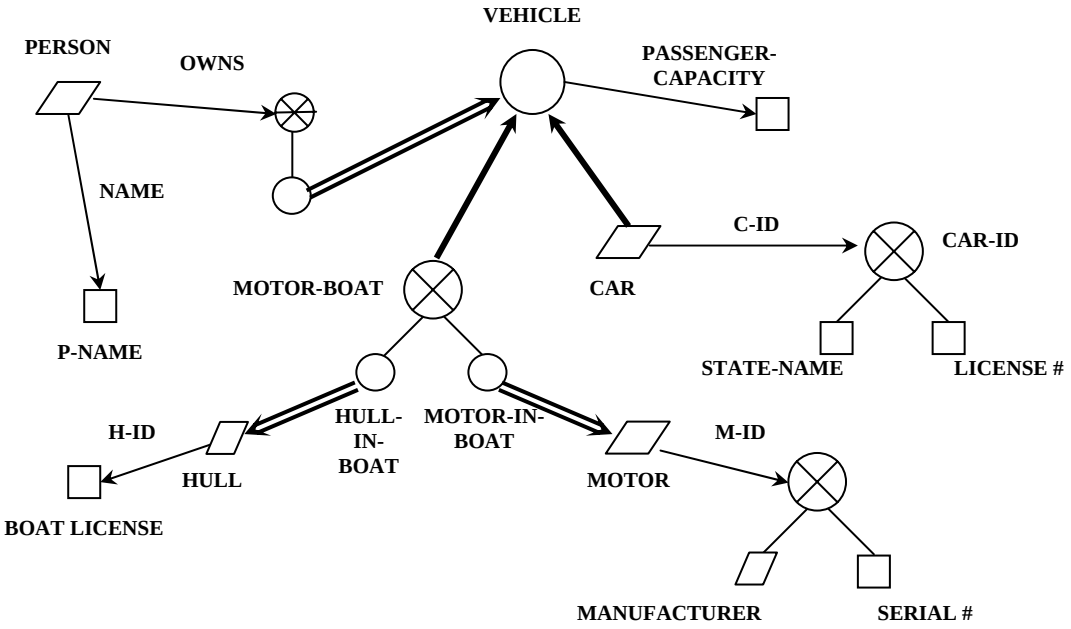
ISA3: For each generalisation edge, the tail is primary and the head is both primary and free.

ISA4: There is no cycle in the object definition graph.

ISA5: Two directed paths of specialisation edges sharing the same origin can be extended to a common vertex.

An *IFO schema* is an IFO graph that satisfies the five rules ISA1 to ISA5.

The following figure (taken from [Abiteboul 1987]) gives a sample IFO schema of a vehicle:




### 2.2.5. Update semantics in the IFO model

Since IFO is one of the first formally defined database models that incorporates essentially arbitrary ISA relationships, functions, and object constructions, it provides a unique framework in which to study updates. In particular, it allows us to carefully examine different ways of *update propagation,* i.e. ways in which a modification of the data in one part of a database can affect data associated with other parts of a database. Generally speaking, IFO provides a strong basis for *mathematical analysis of update operations.*

An update on an object is defined as follows: Let R be a type, *p* a vertex of R, O, O' objects of type R. An *update of R at p* is one of the following triples:

(*p*, O, O'); a *modification* (replace object O by O' at a node *p*)

(*p*, ⊥, O'); an *insertion* (insert O' at *p*)

(*p*, O, ⊥); a *deletion* (delete O at *p*)

Some update operations lead to propagational effects, i.e. they change instances residing in some other vertices. All of the above specified updates are permitted at the root of a type, in which case it has the obvious effect. Second, the update might result from updates elsewhere in the schema, and propagate to one or more leaves of the type. We now consider how updates directed at the leaves of types might arise. Suppose, e.g., that there are two types defined, namely PHONES (a set of valid phone numbers) and CORPORATION (an aggregation of CNAME (printable), OWNER (abstract) and PHONES (a set)). If the phone number 362-0726 is discontinued, which is specified as (PHONE, 362-0726, ⊥), then it

should also be deleted from the instance *I* of CORPORATION. (According to the IFO convention, an instance of a type S is a *finite set* of *dom(S),* rather than a single element of *(S)*.) Thus if the previous instance contained a tuple [Energy Inc., John, {339-3035, 362-0726}], the new instance will contain [Energy Inc., John, {339-3035}]. On the other hand, a deletion of We-Fix Inc., requested by (CNAME, We-Fix Inc., ⊥) would result in the complete removal of the whole We-Fix tuple from *I*.

Insertions are not permitted at types leaves. This is because in general there is no way of knowing which particular objects in the underlying instance should be modified to reflect the requested change. (For instance, which tuples of the overall instance should be affected by an insertion of 724-2115 at the PHONE node?) This assumption implies a fundamental difference between a replacement on the one hand, and a deletion followed by an insertion on the other.

A single update at one part of a schema may result in several simultaneous updates directed at the leaves of some type. For this reason it is important to understand how the updates interact with one another. A formal, recursive definition which captures the intuition of updates 'bubbling' up through a type is given in [Abiteboul 1987]. We conclude this discussion by a fundamental result from this work. It states that updates can be applied to each leaf of a type separately in any order, or that they can be applied all at once:

Let S be a non-atomic type and $L = \{p_1,...,p_n\}$ be the set of leaves of S. Also,

let $M_i = \{(p_i, O, O')$ for some O, O'$\}$ be updates of $p_i$, for $1 \le i \le n$, and let $\sigma$ be a permutation of {1,...,n}. Then for each IFO instance *I* of S it holds:

$M_{\sigma(n)}[M_{\sigma(n-1)}[...[M_{\sigma(1)}[I]]...]] = (\cup_{i=1}^{n} M_i)[I]$.

### *2.3. SDM Model*

Semantic Database Model (SDM, see [Hammer 1981]) is a high-level semantics based database description and structuring formalism for databases. The authors, viz. Michael Hammer and Dennis Mc Leod, intended to develop a database model that would enable a designer to naturally capture much more of the meaning of a database than it is possible with other contemporary (in early 80-ties) database models. SDM has been designed with a number of specific kinds of uses in mind. First, SDM is meant to serve as a *formal specification* mechanism for describing the semantics of a database; an SDM schema provides a *precise documentation* and communication medium for database users. Second, SDM provides the basis for a variety of high-level semantics-based *users interfaces* to a database. Finally, SDM provides a foundation for supporting the effective and *structured design* of databases and database-intensive applications systems. Having in mind the above criteria, which, in fact, every conceptual schema should meet, the authors stated the following essential principles of a database description and structuring formalism:

(1) The constructs of the database model should provide for the explicit specification of a large portion of the *meaning* of a database. The semantic expressiveness of classical record-oriented models is limited; their simple record-like constructs are too close to computer viewing the database and too far from users viewing application environment. There is a need for structural constructs that are highly user oriented and expressive of the application environment.

(2) A database model should support a *relativist view* of the meaning of a database, and allow the structure of a database to support *alternative ways* of looking at the same information. In order to accommodate multiple views of the same data and to make the evolution of new perspectives on the data possible, a database model must support schemata that are *flexible*, potentially *logically redundant*, and integrated. For example, an association between two entities can legitimately be viewed as an attribute of the first entity, as an attribute of the second entity, or as an entity itself. (In the HIT data model we can view it also in a probably most natural way as an n-ary attribute; attributes of SDM are, however, only unary.) A schema should make all three of these interpretations equally natural and direct. Therefore, the conceptual database model must provide a specification mechanism that simultaneously accommodates and integrates these three ways of looking at an assignment. A consequence of this *principle of relativism* is that, in general, the database model should not make rigid distinctions between such concepts as entity, association and attribute. (Note that this is quite a new, revolutionary principle which is not met by most of the other higher-level database models, with the exception of HIT, ORM and COMIC, perhaps, see below. Using the E-R model, e.g., requires the database schema designer to sharply distinguish among these concepts at the very beginning of the design work, and changing the view is very difficult.)

(3) A database model must support the definition of a schema that is based on *abstract entities*. Specifically, this means that a database model must facilitate the description of relevant entities in the application environment, collections of such entities, relationships (associations) among entities, and structural interconnections among the collections. Moreover, the entities themselves must be distinguished from their syntactic identifiers (names); the user-level view of a database should be based on actual entities rather than on artificial entity names.

Following the above described principles the specification of an SDM schema can be characterised as follows:

1) A database is to be viewed as a collection of *entities* that correspond to the actual objects in the application environment.
2) The entities in a database are organised into *classes* (types, sorts in other models) that are meaningful collections of entities.
3) The classes of a database are not in general independent, but rather are logically related by means of *interclass connections* (ISA hierarchies and grouping).
4) Database entities and classes have *attributes* that describe their characteristics and relate them to other database entities. An attribute value may be derived from other values in the database.
5) There are several primitive ways of defining interclass connections and derived attributes, corresponding to the most common types of information redundancy appearing in database applications. These facilities integrate *multiple ways of viewing* the same basic information, and provide building blocks for describing complex attributes and interclass relationships.

### 2.3.1. SDM classes

An SDM database is a collection of entities that are organised into *classes* which are specified by an SDM schema. Classes essentially correspond to types or sorts of other database models, but in contradistinction to most of them SDM does not distinguish between „entity classes“, „value classes“ and „relationship classes“. Each class in an SDM schema has the following features.

(1) A class is identified by its *class name*. Multiple synonymous names are allowed. Each name must be unique with respect to all the class names used in a schema.
(2) The class is a homogeneous collection of its *members*: the entities of one particular type that constitute it. These entities may be:

- Concrete objects, such as books, readers, authors, etc.
- Events, such as a loan
- higher-level entities such as categorisations (e.g. BOOK_TYPES) and aggregations (e.g. DEPARTMENTS) of entities
- values that are syntactic identifiers (strings), such as the class of all the possible names (NAMES) and the class of all the possible calendar dates (DATES).

Following the principle of relativism, an SDM schema does not label a class as containing „concrete objects“ or „events“ or „strings“. No such fixed specification is included in the schema.
(3) An optional textual *class description* describes the meaning and contents of the class.
(4) A class has a collection of *attributes* that describe members of that class or the class as a whole. Hence SDM, similarly as the other models (with the HIT exception, see Section 3.2.3), allows only for unary attributes. There are two types of attributes, classified according to their applicability: *Member attributes* which describe some aspects of each member of a class, and *class attributes* which describe a property of a class taken as a whole.
(5) The class is either a *base class* or a *nonbase class.* A base class is one that is defined independently of all other classes in the schema; it can be thought of as modelling a primitive entity in the application environment, for example, BOOKS. Base classes are mutually disjoint. A nonbase class is one that does not have an independent existence; rather it is defined in terms of one or more other classes. In SDM classes are structurally related by means of *interclass connections.* Each nonbase class has associated with it one interclass connection. If no interclass connection is present with a class in the schema, the class is a base class. There are two main types of interclass connections in SDM: *the subclass connection* and *the grouping connection,* see below.
(6) If the class is a base class, it has an associated list of groups of member attributes; each of these groups serves as a logical key (*identifier*) to uniquely identify the members of the class.
(7) If the class is a base class, it is specified as either *containing duplicates* or *not containing duplicates*. The default is duplicates not allowed; in this case all of the member attributes of the class taken together can serve as a unique identifier.

The interclass connections in SDM are specified in very details. They are the subclass connections and the grouping connections. *The subclass connection* specifies that the members of a nonbase class S are of the same basic entity type as those in the class C to which S is related via the interclass connection. Thus S becomes a subclass of the given class C. The very same entity can thus be a member of many classes. In SDM a subclass S is defined by specifying a class C and a *predicate P* on the members of C; S consists of just those members of C that satisfy P. Several types of predicate are permissible:

- A predicate on the member attribute of C can be used: we get an *attribute-defined subclass*. For example we can define a subclass UNIVERSITY-EDUCATED of the class PERSON by specifying the member attribute predicate „where Education = ‘university’“.
- The predicate „where specified“ can be used to define S as a *user-controllable subclass* of C. In this case the definition of S does not identify which members of C are in S; rather the membership of S is directly and explicitly controlled by users. Taking an example from [Hammer 1981] we can define BANNED-SHIPS as a subclass of SHIPS by a predicate „where specified“ which allows some authority to ban a ship from U.S waters, e.g. Of course, this case might be handled as the previous one, namely by introducing a dummy

member attribute of the parent class whose sole purpose would be to specify whether or not the entity is in the subclass. But this would be a confusing method of capturing semantics of the application environment.

- Another possibility to define a subclass is called a *set-operator-defined subclass*. That is using the intersection, union and difference subclass definition primitives. A class *intersection* capability consists in claiming that members of S are just those members of C that also belong to two other specified subclasses $C_1$ and $C_2$ of C. For instance the class BANNED-OIL-TANKERS can be defined as the subclass of SHIPS that contains members common to the subclasses OIL-TANKERS and BANNED-SHIPS. A *union* subclass S contains those members of C that are either in the subclass $C_1$ or in the subclass $C_2$. For instance class SHIPS-TO-BE-MONITORED can be defined as a subclass of SHIPS with the predicate „where is in BANNED-SHIPS or is in OIL-TANKERS-REQUIRING-INSPECTION“. A *difference* subclass contains those members of C that are not in $C_1$. For example SAFE-SHIPS can be defined as those ships which are not in BANNED-SHIPS.
- The final type of defining a subclass is called an *existence subclass*. In this case the subclass S consists of those members of C that are currently values of some attribute A of another class $C_1$.

*The grouping connection* allows for the definition of a nonbase class, called a *grouping class* G, whose members are of a higher-order entity type than those in the underlying class U. A grouping class is second order, in the sense that its members can themselves be viewed as classes whose members are taken from U. There are again several possibilities of defining a grouping class:

- The grouping class G is defined as consisting of all classes formed by collecting the members of U that have a common value of one or more designated member attributes of U (*an expression defined grouping class*). For example, BOOKS-CATEGORIES can be defined as a grouping type of BOOKS with the grouping expression “on common value of type”. Note that particular members of the grouping class are subclasses of the class U that are attribute defined. Some of them can be actually specified in the schema as subclasses. We might, for instance, specify a subclass BOOKS-ON-PHILOSOPHY of the class BOOKS.
- The second way of defining a grouping class G is to provide a list of classes ($C_1$, $C_2$, ..., $C_n$) that are defined in the schema; these classes are the members of the grouping class G; an *enumerated grouping class.*
- A grouping class G can be defined as a collection of user-controllable subclasses of some underlying class: *a user-controllable grouping class.* For example, class CONVOYS is defined as a grouping of SHIPS ‘as specified’. In this case, no attribute exists to allow the grouping of ships into convoys and particular convoys are not themselves defined as classes in the schema; rather, each member of CONVOYS is a user-controllable group of ships that users may add or delete from.

Entities are application constructs that are directly modelled in an SDM schema. However, there must also be some mechanism to represent these entities — objects of the real world — in the computer database. One cannot enter or display a real entity on a computer terminal, for instance. These representations of entities are called SDM names, and a *name class* in SDM is a collection of strings, namely a subclass of the built-in class STRINGS. Every SDM name class is defined by means of the subclass connection by the predicates specified above. For convenience, particular name classes NUMBERS, INTEGERS, REALS and YES/NO (Boolean) are also built in SDM.

### 2.3.2. SDM attributes

As stated above, each class has an associated collection of attributes. Hence attributes in SDM are, similarly as in other models (with the HIT exception), unary functions. Each attribute of a schema has the following characteristics:

- An attribute *name* identifies the attribute. An attribute name must be unique within a „family“ of classes to which it applies. (This is necessary to support the attribute inheritance.)
- The attribute has a *value* which is either an entity in the database (a member of some class) or a collection of entities. Any class in a schema may be specified to be the value class of an attribute.
- The *applicability* of an attribute is specified, i.e. stating that the attribute is either a *member attribute* or a *class attribute.* A member attribute applies to each member of a class, whereas a class attribute applies to a class as a whole, and has only one value for the class (for instance number of elements).
- An (optional) *attribute description* is a text which specifies the meaning of the attribute.
- The attribute is specified to be either a *single valued* or a *multivalued.* The value of a single valued attribute is a member of the value class, whereas the value of a multivalued attribute is a subclass of the value class.
- An attribute can be specified to be *mandatory*, i.e. a null value is not allowed for it. Identification attributes have to be specified mandatory.
- An attribute can be specified to be *not changeable*, i.e. its value when once set cannot be changed except of an error correction. Identification attributes are usually specified as not changeable.
- A member attribute can be specified to be *exhaustive* of its value class. This means that each member of the value class of the attribute, say A, must be the A value of some entity.
- A multivalued member attribute can be specified to be *non-overlapping,* which means that the values of the attribute for two different entities have no entities in common; that is each member of the value class of the attribute is used at most ones.
- An attribute may be *related* to other attributes, and/or *defined* in terms of the values of other attributes in the schema. In both these cases the attribute is informationally redundant.

The last point needs some explanation. Member attribute interrelationships are *inversion* and *matching.* They are mechanisms for establishing the equivalence of different ways of viewing the same essential relationship among entities. Binary relationship is characterised by a pair of inverse attributes. For instance to express the relationship ‘in what country a ship is registered’ we use two inverse attributes: Ships-registered-here of COUNTRY and Country-of-registry of SHIPS. Another way of relating an attribute to the other attributes is matching which enables us to specify higher degree relationships among entities. Instead of a complicated definition we again provide an example from [Hammer 1981]. Suppose it is necessary to establish a ternary association among oil tankers, countries and dates, to indicate that a given tanker was inspected in a specified country on a particular date. (Note that in other models this situation would be handled as a binary relationship between entities tanker and country with the attribute date of an inspection.) In SDM we define a class COUNTRY-INSPECTIONS with three attributes: Tanker-inspected, Country-of-inspection, Date-inspected. These attributes would then be matched with the appropriate attributes of OIL-TANKERS, COUNTRIES and DATES that also express this information. The combined use of matching and inversion allows an SDM schema to accommodate *relative viewpoints of an association.* For instance, one may view the ternary relationship in

the above example as an inspection entity (a member of the class COUNTRY-INSPECTIONS), or as a collection of attributes of the entities that participate in the association. Similarly, a binary relationship defined as a pair of inverse attributes could also be view as an association entity, with matching used to relate that entity to the relevant attributes of the associated entities. We can see that viewing relationships in SDM is much more flexible than that of the, e.g., E-R model, though at the costs of storing much redundant information in the schema.

Another way in which attributes may be related to the other attributes is the so called *derivation*: an attribute is defined whose values are calculated from other information in the database. Such an attribute is called *derived* and the specification of its computation is its derivation. There is a small vocabulary of attribute derivation primitives that directly model the most common types of derived information. They are, e.g., ordering, existence attribute, recursive tracing, contents, subvalue, intersection, union, difference, arithmetic expressions, maximum, minimum, average, sum, etc. Analogous primitives can be defined for class attribute derivations.

The last notable feature of SDM is automatic *attribute inheritance.* Generally speaking, a subclass S of a class C inherits all the *member attributes* of C. *Class attributes* describe properties of a class taken as a whole and so are *not inherited* by a subclass. In order for an attribute to be inherited from class C by class S, both its meaning and its value must be the same for C and S. This is not true for class attributes. Although a subclass may have a similar class attribute to that defined for its parents class, for example 'An-average-age-of-members', their values will in general not be equal. Two special cases must be mentioned: When a subclass S is defined as an intersection of classes $C_1$ and $C_2$, it inherits all the member attributes of $C_1$ and all the member attributes of $C_2$. On the other hand, a class S defined as the union of classes $C_1$ and $C_2$ inherits all the member attributes shared by $C_1$ and $C_2$. The rules of inheritance need not be explicitly applied by an SDM user; they are integral part of SDM and are automatically applied wherever appropriate.

Concluding this section we can state that SDM is in many ways analogous to a number of other contemporary semantic database models, to name at least [Chen 1976], [Codd 1979], [Mylopoulos 1978], [Shipman 1981], [Smith 1979], [Su 1979]. Where SDM differs from these is in its emphasis on relativism, flexibility and redundancy. SDM schema supports multiple ways of viewing the same information, since different users may have different slants on the database and even a single user's perspective may evolve in time. Consequently, redundant information plays an important role in an SDM schema. Yet another aspect is connected with this fact: data manipulation facilities were not mentioned, since SDM is based on the duality principle between schema and procedure. From this perspective, *any query against the database* can be seen as a reference to a particular *virtual data item*. Whether that item can easily be accessed in the database, or whether it can only be obtained by means of the application of a number of complicated manipulation operations, depends on what information has been included into the schema by a designer. Frequently retrieved data items would most likely be present in the schema, often as derived data, while less commonly requested information would have to be dynamically computed. Anyway, taking into account these features of SDM, we would say that SDM schema is from a general point of view rather an external schema (or view) than a conceptual schema. In our opinion, redundant information should not be included into a conceptual schema unless there are some special reasons for that (efficiency, reliability).

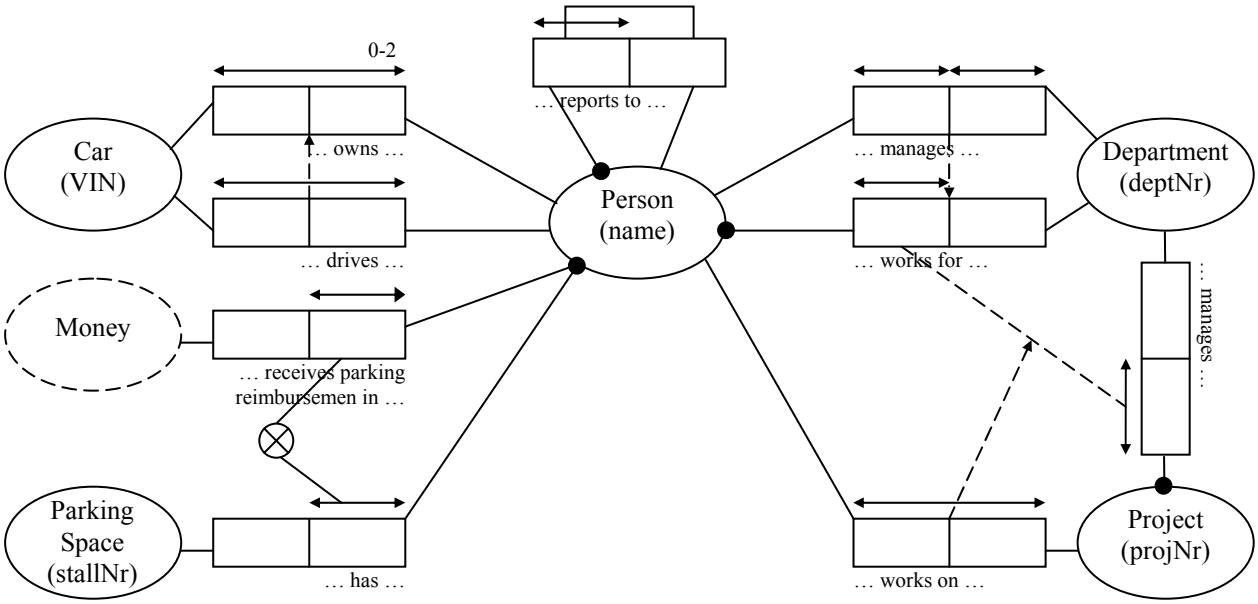
### *2.4. ORM modelling*

ORM originated in the mid-1970s as a semantic modelling method, one of the early versions being NIAM (Natural language information analysis method), and has since been extensively revised by many researchers. According to the ORM methodology, designing a database requires a complete understanding of the subject area, or universe of discourse (UoD), to be implemented. Thus a good database model is one that specifies the Uod in a clear and unambiguous way. ORM uses a natural language and easy to understand diagrams that are populated with example data to accomplish this goal. Another notable aspect of ORM is that, since it is based on a natural language, it can be completely expressed in either graphical or textual format. We shall see that these features are shared by the HIT data model as well. But unlike the HIT data model which is well theoretically founded, we miss in ORM precise theoretical-logical background. It is said [Becker 1998a] that ORM is vastly superior to the Chen's E-R model. Below we try to examine whether such a claim is justified.

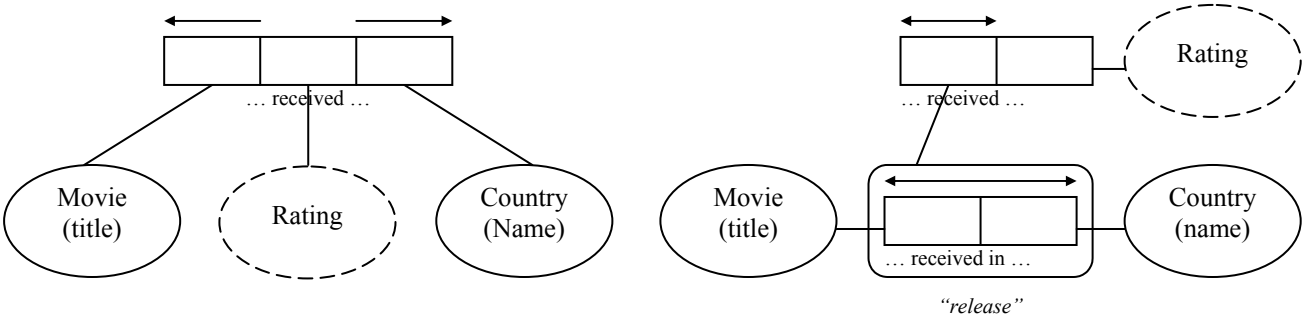
The root of ORM is an elementary fact. You express the UoD in terms of *objects* (such as person, department, project, etc.) playing *roles* (works for, manages, reports to, etc.), and traditionally express all information in terms of *elementary facts, constraints* and *derivation rules*. In contradistinction to the E-R model, you make no distinctions whether an object is an attribute or an entity. ORM uses a natural language and easy to understand diagrams that may be populated with example data. Further, similarly as in the HIT data model (see Chapter 3.), a natural language is much easier for your users to understand, express, and verify, than a technical jargon that is often tended to use.

Using ORM, the designer just expresses the UoD in simple, easy to understand facts such as: „Person works for Department", „Person works on Project", „Person manages Department", „Department manages Project", „Person reports to Person", „Person has Parking Space", „Person receives parking reimbursement in Amount", „Person drives Car", „Person owns Car". Using this fact based approach, ORM makes reengineering and schema evolution quite simple. Further, this approach simplifies normalisation worries: the elementary nature of the facts ensures that the schema is in an optimal (usually 4th) normal form. This approach allows to make (in E-R terms) attribute level constraints. What follows is an example of an ORM schema [Becker 1998c] (sample data omitted) illustrating the above facts and the following constraints:

- Subset constraint: Ä Person can manage a Department only if that Person works for that Department".
- Equality constraint: „If a Person owns a Car she must also drive that Car, and if a Person drives a Car she must also own that Car".
- Exclusionary constraint: „A Person can either have a Parking Space or receive a parking reimbursement, not both".
- Mandatory disjunction: „A Person must either have a Parking Space or receive a parking reimbursement".
- Frequency: „Person may own a maximum of two Cars".
- Ring constraint: Given a fact in the form of 'Person plays a role with a Person', specify if the relationship is reflexive, symmetric, transitive, irreflexive, asymmetric, antisymmetric, and/or intransitive, such as: "A Person cannot report to themselves (irreflexive)".
- Or any combination of the above constraints such as: "A Person can only work on a Project if that Person's Department manages that Project".




ORM also allows you to have facts with an arity (number of roles in the fact) greater than two. For example, let's say you have fact like „Movie receives Rating in Country". ORM allows you to model this fact naturally or via nesting, which allows you to add other (potentially optional) roles to the nested fact. This situation is illustrated by the following figure[Becker 1998c]:

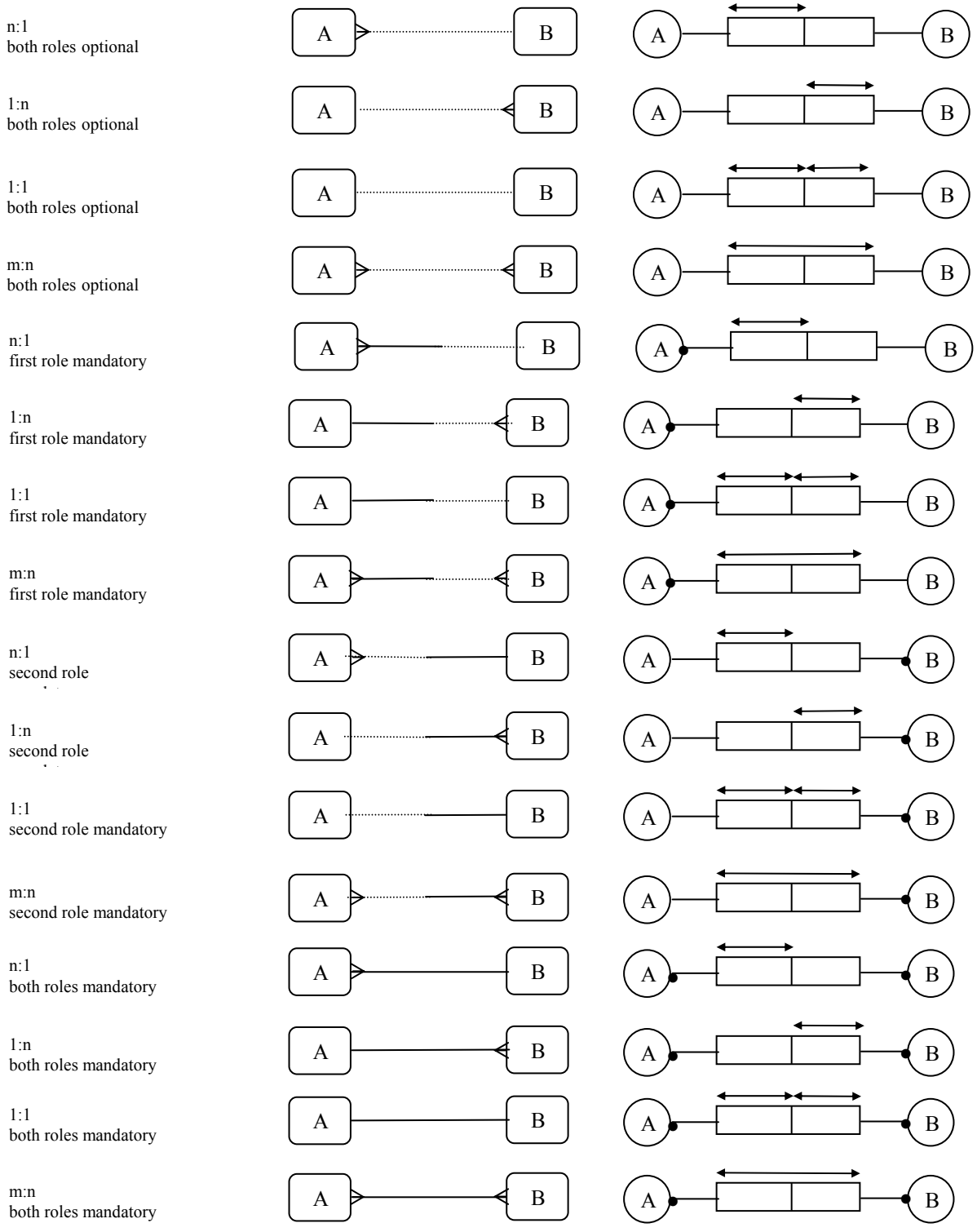



ORM also uses an accurate, complete definition of subtyping and inheritance, derived fields (including a distinction between merely derived and derived-and-stored), and schema transformation and evolution. Finally, mapping the conceptual schema into a logical schema, a physical schema, and implementing the constraints are often trivial (with the use of CASE tools). However, mapping issues and CASE tool implementation are beyond the scope of this work.

**A quick explanation of the ORM symbols used.**

*Objects* are shown as ellipses. The object name is denoted inside of the ellipse. Sometimes, an additional information is included in parenthesis below the object name. This is the so called reference mode, and serves to identify an object. For example, Person object has a reference mode of 'name', Car object has a reference mode VIN. Reference mode can be attached only to entity objects (denoted with a solid ellipse). Entity objects are conceptual things like Person, Car, Department. Value objects (shown as a dashed ellipse) are merely values (instances of data) such as a string, a date, or a number.

*Predicates,* or the connections between objects, are shown as boxes connected to the objects with an additional text information (such as ‘owns’, ‘drives’, ‘manages’, etc.) denoted below the boxes. The dots shown on the connection between an object and a predicate mean that the role is mandatory (for example, every Person must work for a Department, every Project must be managed by a Department). Bold arrows pointing between objects (which are not shown in our figures, but their semantics is intuitively clear) denote supertypes and subtypes. The arrow always points from subtype to supertype. The tipped arrows (or bars) above the predicate boxes show („internal“) uniqueness. To explain the role of bars, we show the sixteen possible patterns of uniqueness and mandatory role constraints in the well-known Oracle E-R notation and ORM notation:



**Advantages of using Object Role Modelling**

E-R's conceptual model allows you to view objects and relationships (and a few constraints) at a high level in a compact notation. However, E-R's use of (descriptive only) attributes makes the model inherently unstable with regard to schema changes. Further, E-R schemas make it difficult to apply a population check (with real data) and are missing many important constraints, particularly at the attribute level. E-R relationships also tend to be binary (in Oracle notation only binary ones are allowed, while ORM allows relationships of any arity), which forces you to use unnatural intersection entities and other conceptual falsehoods. ORM allows you to speak to business experts in their own language, without having to use such artificial constructs.

Other benefits of ORM include:

- The fact-based approach of ORM is a simpler and more accurate approach as it is easier to get one fact correct than many facts simultaneously.
- E-R tends to set a level of importance (is it an entity or an attribute?) early in the modelling process. If you do not perform those initial steps correctly the first time, you will end up changing your model later (and possibly correcting the data itself). ORM sets no initial importance to objects at all. Rather, importance of a particular fact will reveal itself much later on (by discovering, e.g., that you have many roles attached to an object).
- In ORM, semantic domains (i.e. units or ranges such as 'name', 'SSN', 'age', 'date') are automatically included. This allows for stronger typing and is less error prone.
- ORM has a less implicit duplication of attributes than E-R does. For example, in E-R, you could have an athlete entity with two attributes, the country they represent and their birthplace (experienced modellers would point out that this is bad E-R modelling but nothing prevents you in E-R from doing so). For the sake of argument, say the initial model only had country the athlete represents and the birthplace was added later. Making this change is not so easy once the model is implemented. In ORM, you would have only two objects, Athlete and Country which play two roles with each other: '... represents ...' and '...born in ...'. Again, ORM is less prone to errors (the previous mistake is impossible). Less error prone means your model (and therefore, your database) is much more stable and the magnitude of problems caused by a change to the model is lessened.
- ORM has many more constructs inherent to the language, and is therefore, more expressive of the actual UoD.
- In ORM, you get a normalised database at no extra charge. ORM's use of the elementary fact insures no functional dependencies will violate normalisation (1NF, 2NF, 3NF, BCNF, EKNF, and 4NF, see Section 5.3). And ORM's rich constraint implementation will allow you to capture many constraints typically considered to be 5NF considerations (semantic rules) by E-R modellers.
- Since business requirements are subject to ungoing change, it is critical that the underlying data model be crafted in a way that minimises the impact of these changes. The ORM framework is more stable under business changes than E-R model, and facilitates the remaining changes that need to be made. This stability applies not only to the model itself, but also to conceptual queries based on the model.

We devoted much attention to the ORM modelling, maybe rather neglecting other semantic data models, for, in our opinion, all the above advantages of ORM make the model

in a way superior to the others. Moreover, as it will be shown in the following chapters, all these advantages are shared by the HIT data model as well (the description of which is a core of this work). Why then do we advocate for another (HIT) data model? The reason is that one of the goals of this work is to submit theoretical-logical background of conceptual modelling, to provide precise definitions and explication of all the constructs used, particular constraints, etc. And it is just the HIT data model (and probably the only one, as far as we know) that provides such a precise underlying logical apparatus.

### *2.5. UML data model*

In this work we actually do not deal with the object-oriented (O-O) approach that has become very popular in the last decades, for, in our opinion, there is no such thing as object-oriented conceptual analysis. There is only O-O design and implementation. The only benefit that could be regarded as being brought to the conceptual analysis phase from the O-O approach is the change of *system decomposition criteria.* System decomposition has been traditionally considered as creating a great problem. When the system in question is large and complicated, it cannot be analysed and handled in its whole. We have to decompose it first into subsystems of a reasonable size which can be afterwards analysed in details. But then there is a question: which criterion of decomposition to choose? A „functional one“ or a „data one“? A long time ago, before the O-O approach appeared and became popular, functional criterion had been preferred. The consequences had often been nearly fatal: there were so many links, relationships between particular subsystems (since one and the same object could have been mirrored in many subsystems) that the whole system could actually not be handled. Moreover, the system was not adaptable to changes because any change of functions (which are very frequent) could cause a change of decomposition, the system was unstable.

The above problem has been probably definitely solved by the O-O approach: when decomposing a system, concentrate on the main objects of interest and assign the respective functions to them. The main objects of interest are stable unlike the functions that the system should perform (above these objects). Moreover, when decomposing according to objects, the resulting subsystems are relatively independent and the whole system is easy to operate.

So much for the O-O approach. Nevertheless, we now briefly describe the family of UML (Universal Modelling Language) that has been considered to be an O-O design tool. It can be shown [Schewe 2000] that in many respects UML is far from being new: With respect to syntax it just re-invents many of the old ISOTEC (Integrated Software Technology [EDV 1983]) constructs and introduces new names for them. With respect to semantics it does not present precise semantic definitions. If these are added, the limitations of the expressiveness of the UML become apparent. Quoting from [Schewe 2000]: *the UML is the modern winner in terms of its lack of precise definitions, lack of clear semantics, lack of clarity with respect to abstraction levels, and lack of pragmatic methodology.* Referring for details to [Booch 1999], [Rumbaugh 1999], we will concentrate on UML data modelling capabilities and describe it from an ORM perspective [Halpin 1998].

UML includes diagrams for use cases, static structures (class and object diagrams), behaviour (state-chart, activity, sequence and collaboration diagrams) and implementation (component and deployment diagrams). For data modelling purposes UML uses *class diagrams*, to which constraints in a textual language may be added. Although class diagrams may include implementation detail (e.g. navigation and visibility indicators), it is possible to use them for analysis by omitting such detail. When used in this way, class diagrams essentially provide an extended Entity Relationship notation.

UML's object-oriented approach facilitates the transition to object-oriented code, but can make it awkward to capture and validate business rules with domain experts. This problem can be remedied by using an ORM's fact-oriented approach where communication takes place in simple sentences, and each sentence type can easily be populated with multiple instances. ORM harmonises well with UML, since both approaches provide direct support for roles, n-ary associations and objectified associations. To better exploit the benefits of UML, or E-R for that matter, ORM can be used for the conceptual analysis of business rules, and the resulting ORM model can be easily transformed into a UML class diagram or E-R diagram.

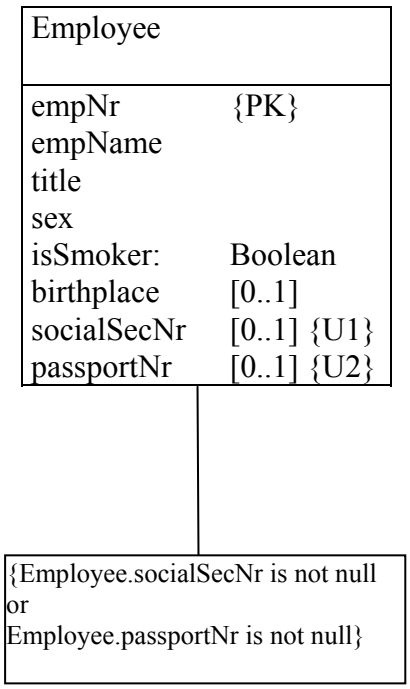
UML classifies *instances* into *objects* and *data values*. UML objects basically correspond to ORM entities, but are assumed to be identified by *oids* (hidden, system generated object identifiers). For analysis purposes however, we need to ensure that humans have a way of identifying objects in their normal communication. UML data values basically correspond to ORM values: they are constants (e.g. character strings or numbers) and hence require no oids to establish their identity. Entity types in UML are called *classes*, and value types are called *data types*. Note that „object" means „object instance", not „object type". A relationship instance in UML is called a *link,* and a relationship type is called an *association.*

Because of reliance on oids, UML does not require entities to have a value-based reference scheme. This can make it impossible to communicate naturally at the instance level, and ignores the real world database application requirement that humans have a verbal way of identifying objects. It is important therefore to include value-based reference in any UML class diagram intended to capture all the conceptual semantics about a class.

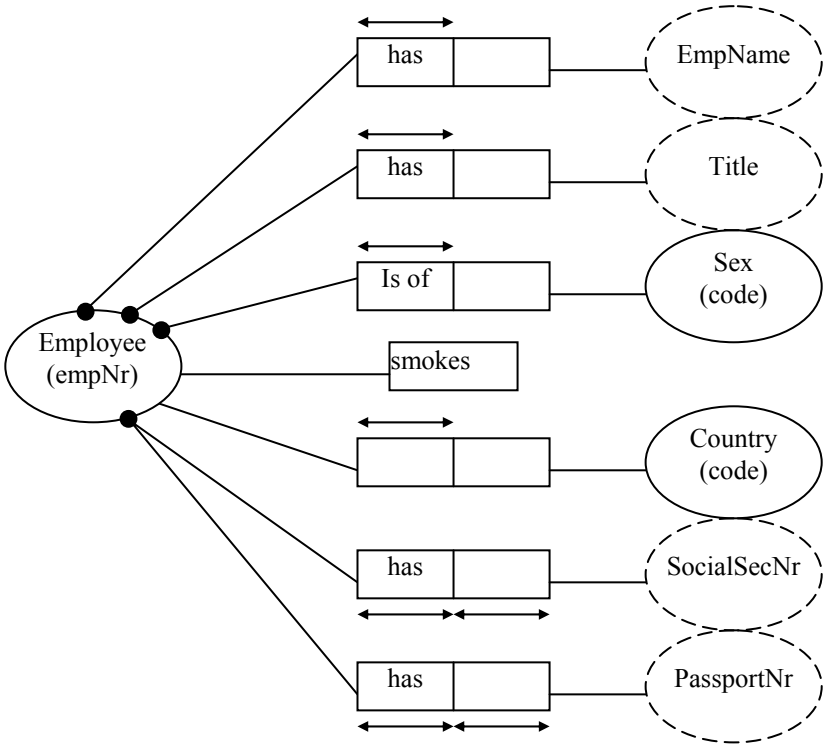
Like other E-R notations, UML allows relationships to be modelled as (descriptive) *attributes.* For instance, in the following figure (a) [Halpin 1998] the Employee class has eight attributes. Classes in UML are depicted as a named rectangle, optionally including other compartments for attributes and operations. The corresponding ORM diagram is shown in the following figure (b). True to its name, ORM models the world in terms of just objects and roles, and hence has only one data structure - the relationship type. This is one of the fundamental differences between ORM and UML (and E-R for that matter). *Wherever an attribute is used in UML, OML uses a relationship instead.* As a consequence, ORM diagrams typically take up more room than corresponding UML or E-R diagrams. But this is a small price to pay for the resulting benefits.

Here is an e*xample* illustrating the difference between a description of an employee using an ***E-R schema —*** Fig (a) and ***UML diagram —*** Fig (b):

(a)




(b)




***E-R schema*** ***UML diagram***

As stated above, the ORM model indicates that employees are identified by their employee numbers. The top three mandatory constraints indicate that every employee in the database must have a name, title and sex. The other black dot where two roles connect is a *disjunctive mandatory role constraint,* indicating that the disjunction of these roles is mandatory (each employee has a social security number or a passport number, or both). Although each of these two roles is individually optional, at least one of them must be played.

In UML, attributes are mandatory by default. In the ORM model, the unary predicate 'smokes' is optional (not everybody has to smoke, of course). UML does not support unary relationships, so it models this instead as the Boolean attribute 'isSmoker'. In UML the domain of any attribute may optionally be displayed after it (preceded by a colon). In this example, we showed the domain only for the isSmoker attribute. The ORM model also indicates that Sex and Country are identified by codes (rather than names, say). We could convey some of this detail in the UML diagram by appending domain names (e.g. 'Sexcode', 'Countrycode') after 'sex' and 'birthplace', but these are essentially rather syntactic than semantic domains.

In the ORM model it is optional whether we record birthplace, social security number or passport number. This is captured in UML by appending [0..1] after the attribute name. This is an example of an *attribute multiplicity* constraint. UML does not have a graphic notation for disjunctive mandatory roles, so this kind of constraint needs to be expressed textually in an attached note. Such textual constraints may be expressed informally, or in some formal language interpretable by a tool. In the latter case, the constraint is placed in braces. Although UML provides the Object Constraint Language (OCL) for this purpose, it does not mandate its use, allowing users to pick their own language (even programming

code). This of course weakens the portability of the model. Moreover, the readability of the constraint is typically poor compared with the ORM verbalisation (each Employee has a SocialSecNr or has a PassportNr).

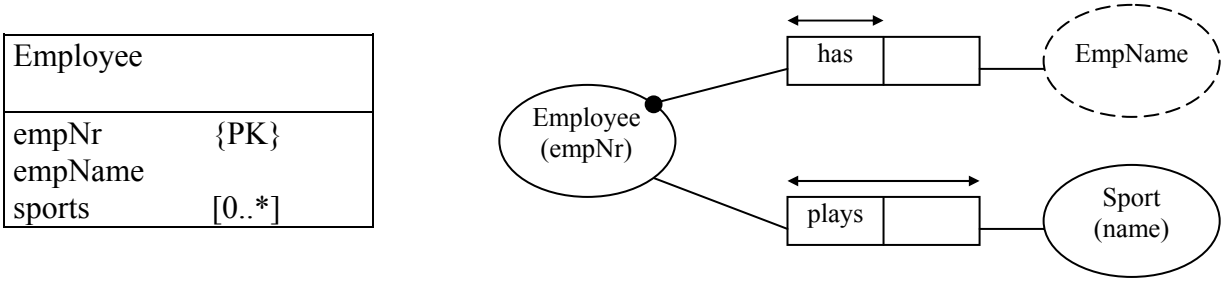
The *uniqueness constraints* over the left-hand roles in the ORM model (including the empNr reference scheme shown explicitly earlier) indicate that each employee has at most one employee number, employee name, title, sex, country of birth, social security number and passport number. Unary predicates have an implicit uniqueness constraint; so each employee instantiates the smokers role at most once (for any given state of the database, or, as we would say, for any state of the world). All these uniqueness constraints are implicitly captured in the UML model, where attributes are single-valued by default.

The uniqueness constraints on the right-hand roles indicate that each employee number, social security number and passport number refers to at most one employee. UML does not have a standard notation for these *attribute uniqueness constraints*. We have chosen here our own notation for this, appending textual constraints in braces after the attribute names (PK = primary key, U = unique, with numbers appended to disambiguate cases where the same constraint might apply to a combination of attributes). The use of 'PK' does not imply the model must be implemented in a relational database using value-based keys; it merely indicates a primary identification scheme that may be used in human communication. Because UML does not provide standard notation for such constraints, and it leaves it up to the modeller whether such constraints are specified, it is perhaps not surprising that many UML models one encounters in practice simply leave such constraints out.

Now that we have seen how single-valued (descriptive) attributes are modelled in UML (and E-R, as the case may be), let's briefly see why ORM refuses to use them in its base modelling, expressing them as (functional) relationships between basic objects (in an analogous way as the HIT data model does, see Ch. 3.2). The reasons can be summarised as follows: modelling stability, easier verbalisation in natural language sentences, highlighting connectedness through semantic domains, easier specification of constraints. Let's illustrate the main one, namely *semantic stability*. ORM models and queries are more stable, because they are free of changes caused by data types evolving into entity types or relationship types, or vice versa. Consider the ORM fact type (discussed above): Employee was born in Country. In E-R and O-O approaches we might model this using a birthplace attribute. If we later decide to record, e.g., the population of a country, then we need to introduce Country as an entity type. In UML, the connection between birthplace and Country is now unclear. Partly to clarify this connection, we would probably reformulate our birthplace attribute as an association between Employee and Country. This is a significant change to our model. Moreover, any object-based queries or code that referenced the birthplace attribute would also need to be reformulated. A typical counter-argument is this: "Good E-R or O-O modellers would declare country as an object type in the first place, anticipating the need to later record something about it; on the other hand, features such as the title and sex of a person clearly are things that will never have other properties, and hence are best modelled by (descriptive) attributes". This attempted rebuttal is flawed. There is nothing in the E-R or O-O models that would prevent the designer from such mistakes that can be in more complicated systems quite frequent, and, in general, you can't be sure about what kinds of information you might want to record later. Moreover, formulation and reformulation of constraints and queries is much easier in ORM than in E-R and UML. Elementary ORM facts are the fundamental conceptual units of information, are uniformly represented as relationships, and how they are grouped into structures is not a conceptual issue. Though outrunning our exposition, we have to state in this place that all the above features of ORM are fully captured by the HIT data model,

where ORM facts are mirrored by HIT-attributes (empirical functional relationships between basic types), which are uniform basic modelling structures, not distinguishing between (descriptive) attributes and relationships as the E-R and other models do.

Multi-valued attributes are in UML recorded by specifying a [0..*] constraint, as shown in the following example. Suppose that we are interested in recording the names of employees, as well as the sports they play (if any). In ORM, this is shown by making the uniqueness constraint span both roles. Since an employee may play many sports, and a sport may be played by many employees, Plays is a many to many (m:n) relationship type. One way of modelling the same situation in UML is shown in the following figure:

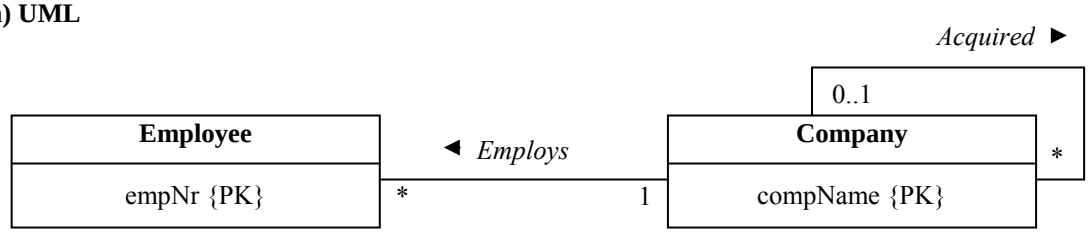



Here the information about who plays what sport is modelled as the *multi-valued attribute* 'sports'. The [0..*] appended to this attribute is a *multiplicity constraint* indicating how many sports may be entered for each employee. The sign '0' indicates that it is possible that no sports might be entered for some employee. Unfortunately, the UML standard uses a *null value* for this case, just like the relational model. The sign '*' indicates that there is no upper bound on the number of sports of a single employee.

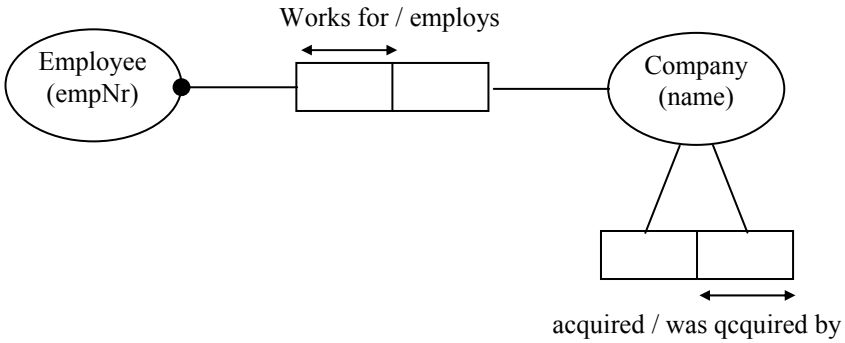
UML gives us the choice of modelling a feature as an attribute or an association (similar to an ORM relationship type). At least for conceptual analysis and querying, explicit associations usually have many advantages over attributes, especially multi-valued attributes. (However, do not confuse the notion of a 'normal' descriptive attribute, as known from current data models with the notion of HIT-attribute that models associations as functional relationships between basic types. See Ch. 3.2). The choice of associations helps us verbalise, visualise and populate associations. It also enables us to express various constraints involving the role played by the attribute in standard notation, rather than resorting to some non-standard extension (as it was done in the above example with braced comments). Another reason for favouring associations over attributes is stability. If we ever want to talk about a relationship, we have to make an object out of it first to attach the new details to it. If we modelled the feature as an attribute, we would not be able to add the details without first changing the original schema: in effect we would need to first replace the attribute by an association. For example consider an ORM fact (association) Employee plays Sport. If we now want to record a skill level for this play, we can simply objectify this association as Play, and attach the fact type: Play has SkillLevel. A similar move can be made in UML if the play feature has been modelled as an association. In the above example, however, this feature has been modelled as the sports attribute; so this attribute needs to be removed and replaced by the equivalent association before we can add the new details about skill level. Hence we have to discuss the notion of ORM objectified relationship types or UML association classes now.

Before discussing UML associations, we review the basic ideas discussed previously. Attributes in UML are depicted as relationship types in ORM. A relationship instance in ORM is called a link in UML (e.g. Employee 101 works for Company 'Visio'). A relationship type in ORM is called an association in UML (e.g. Employee works for Company). In both UML and ORM, a role is a part played in a relationship. The number of roles in a relationship is its arity. ORM allows relationships of any arity. Each relationship type has at least one reading or predicate name. An n-ary relationship may have up to *n* readings (predicate names) to provide natural verbalisation of constraints and navigation paths in any direction. UML uses Boolean attributes instead of unary relationships, but allows relationships of all other arities. Each association may be given at most one name, and this is optional. Association names are normally shown in italics, starting with a capital letter. Binary associations are depicted as lines between classes. Association lines may include elbows to assist with layout or when needed (e.g. for ring relationships). Association roles appear simply as line ends instead of boxes, but may optionally be given role names. Once added, role names may not be suppressed. Verbalisation into sentences is possible *only for infix binaries*, and then only by naming the association with a predicate name (e.g. 'Employs') and using an optional marker '>' to denote the direction. The following figure [Halpin1998] depicts binary associations in both UML and ORM.

**a) UML**




**b) ORM**

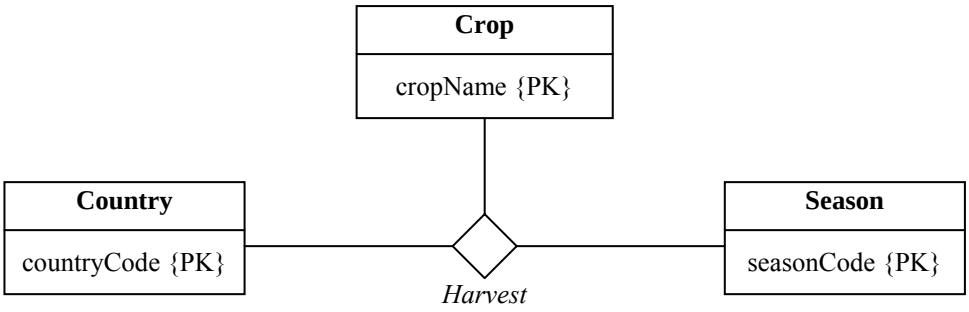



The uniqueness and mandatory role constraints are written beside the relevant roles in a similar way as on attributes. The sign '*' abbreviates '0..*', meaning zero or more; '1' abbreviates '1..1', meaning exactly one, and '0..1' means at most one. Unlike some E-R notations, UML places each constraint on the 'far role', in the direction in which the association is read. Hence the constraints in this example mean: Each company employs zero or more employees; each employee is employed by exactly one company; each company acquired zero or more companies; and each company was acquired by at most one company.

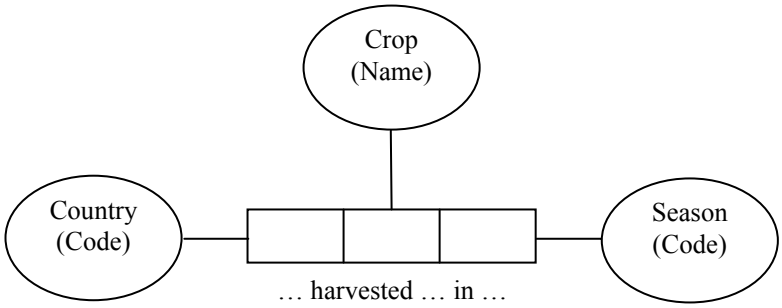
Ternary and higher arity associations in UML are depicted as a diamond connected by lines to classes. Because many lines are used to denote the association, directional

verbalisation is ruled out, so the diagram *cannot be used to communicate in terms of sentences*. The following figure illustrates a ternary relationship expressed both in UML and ORM (constraints omitted).

**(a) UML**

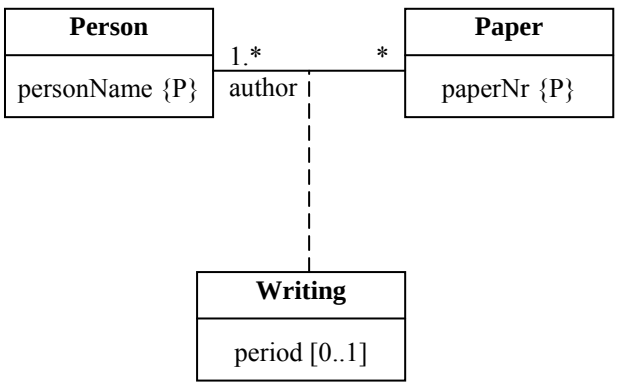



**(b) ORM**

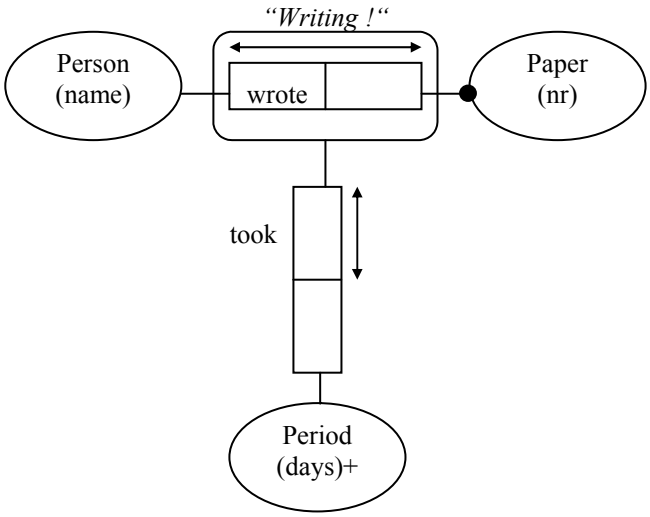



Unlike many E-R versions, both UML and ORM allow associations to be objectified as class object types, called associations classes in UML and nested object types (or objectified relationship types) in ORM. UML requires the same name to be used for the original association and the association class, impeding natural verbalisation of at least one of these constructs. In contrast, ORM nesting is based on linguistic nominalisation (a verb phrase is objectified by a noun phrase), thus allowing both to be verbalised naturally, with different names for each. The following figure depicts an objectified association in UML and ORM:

**UML**




**ORM**



Each person may write many papers, and each paper is written by at least one person. In the UML depiction, we have used '{P}' to indicate the primary reference attributes used for human communication about persons and papers. Since autorship is m:n, the association class Writing has a primary reference scheme based on the combination of person and paper. The optional period attribute stores how long that person took to write that paper. Instead of distancing the objectified association from its underlying association, ORM intuitively envelops the association with an object type frame. Writing is marked independent (displayed with '!') to indicate that a writing object may exist, independently of whether we record its period. ORM displays Period as an object type, not an attribute, and includes its unit.

We are not going to discuss other modelling possibilities, like, e.g., constraints. Just briefly: qualified associations and or–associations. Simple cases where ORM uses an external uniqueness constraint for co-referencing can also be modelled in UML using qualified associations. Qualifier in UML is a set of one or more attributes, whose values can be used to partition the class, and is depicted as a rectangular box enclosing its attributes. For instance the attribute accountNr can be used as a qualifier on the association Bank – Person, effectively partitioning each bank into different accounts. The term or–association is used by UML for one of many associations stemming from a class, where at any given time each class member may participate in at most one of these associations. To indicate this, UML uses what it calls an or–constraint between the associations, attaching the constraint string '{or}' to a dotted line connecting the relevant associations. For instance an account is used by a person or by a corporation. UML's use of 'or' for this constraint is confusing because it is used in an exclusive instead of inclusive sense (in contrast to virtually all computer languages). Now we covered all the high level data structures that can be specified in graphic notation of ORM data models and UML class diagrams. We will not discuss more advanced graphic constraints in ORM and UML, such as subtyping, derivation rules and queries, since they are essentially the same as in other semantic data models, and the technical issues are not interesting. The following table summarises the differences between the two modelling methods with respect to terms and graphic notations for data instances and structures.

| **ORM** | **UML** |
|---|---|
| Entity | Object |
| Value | Data value |
| Object | Object or Data value |
| Entity type | Class |
| Value type | Data type |
| Object type | Class or Data type |
| use relationship type | Attribute |
| Unary relationship type | use Boolean attribute |
| n-ary relationship type | Association |
| n-ary relationship instance | Link |
| Nested object type | Association class |
| Co-reference | Qualified association |

# 3. Conceptual Modelling.

In this chapter we are going to introduce to approaches to conceptual modelling: Finnish and Czech schools. Finnish school is revolutionary in its approach to data modelling. It repudiates all the classical modelling constructs, such as an 'entity set', a 'value set', a 'relationship set', an 'attribute', considering this classification to be a "superimposed structural scheme into which knowledge is often forced" (see [Kangassalo 1993]). It is an interesting attempt to model a given part of reality only in terms of concepts and the relation of the intensional containment between them. Nevertheless, some traditional constructs are hidden here in the form of the so-called definitions, such as generalisation, specialisation, aggregation, attribute. From the theoretical point of view, it is based on Kauppi's concept theory [Kauppi 1967] which can be characterised as a modern elaboration of the traditional set-theoretical doctrine. Surprisingly ignoring the category of possible worlds, it has no tool for the distinction between empirical and analytical notions.

On the other hand, Czech school with its HIT data model (HIT is an acronym for Homogeneous, Integrated, Type-oriented [Zlatuška 1986]) is inspired by classical semantic data models (a good survey can be found, e.g., in [Hull 1987]), to name at least Chen's E-R model [Chen 1976] or an object-function model [Scholl 1990]. It is based on the concept of the 'HIT attribute' which is a generalisation and an exact explication of the traditional modelling construct. Being conceived as an n-ary empirical function, it makes it possible to use a functional approach and to exploit the apparatus of a modified version of the typed λ-calculus. From the logical point of view, it is based on the transparent intensional logic - TIL [Tichý 1988], making use of its possible world semantics (the distinction between empirical and analytical notions is of a key importance here), and of its theory of constructions. Last but not least, a new non-traditional theory of concepts (viewed as some abstract procedures) [Materna 1998] is exploited here.

## *3.1. The Finnish school: COMIC model*

We first briefly reproduce the basic works of H. Kangassalo [Kangassalo 1993], [Kangassalo 1998], and only afterwards, the results of the related works [Niemi 1998], [Niemi 1999], [Junkkari 1999], [Palomäki 1997], [Niinimäki 1999] and [Nilsson 1998] are summarised. When reproducing the ideas of H. Kangassalo and his followers, we simultaneously ask questions and state problems connected with them without great ambitions to answer them. The following text is not a smooth text. It is rather a summary and depicting of the main ideas and claims, with our emphasising the probably problematic places. Our comments are included in curly brackets. We hope that the answers are provided in the Section 3.3 and concluding Section 3.4.

### 3.1.1. COMIC system

The first and basic claim is the following [Kangassalo 1993]: The design of an IS consists in the definition of the borders of an initial Universe of Discourse (UoD), and the development of a conceptual schema of the UoD. In the later work [Kangassalo 1998] this thesis is even strengthened: We should replace the whole information system with the conceptual schema of the UoD, supported with the facilities of manipulating data *corresponding* to the conceptual schema. A conceptual schema defines a systematic 'theory' of the Uod. The concepts are *constructed* on the basis of goals and business rules of the user organisation.

{The question arises here: How are the concepts *constructed?* And even more fundamental question*: What* are the concepts?} The answer provided here is: *A concept* is defined to be an independently identifiable *structured construct* composed of knowledge primitives and/or other concepts. {Well, but what are the knowledge primitives, and in what way do they differ from concepts?} We can read that: knowledge primitives are the smallest structural units of knowledge. The following are some of the most common knowledge primitives:

- *Name* of the concept
- *Intensional relationship* is a relationship between two *concept*. The most important intensional relationship is that of *intensional containment.*
- *Extensional relationship* is a relationship between occurrences of concepts.
- *Identifying property* is a property of concept B intensionally contained in concept A that enables an occurrence of concept B to be used to identify an occurrence of concept A.
- *Condition* is a truth-valued proposition that can be either true or false for a given occurrence of a concept.
- *Constraint* is a truth-valued proposition that must be true for the occurrence of a concept if the constraint appears in the intension of that concept.
- *Conditional constraint* is a truth-valued proposition built up of a set of conditions and a set of constraints. For the occurrence of a concept containing a conditional constraint all constraints must be true when the conditions are true.
- *Value set* is a set of other concepts and their representations associated with a given concept. A value set of atomic values can be associated only with a *basic concept* or a *magnitude concept*. Values in the value set of other concepts, i.e. *derived concepts*, are composed from other values in a way reflecting the structure of the derived concept.
- *Function* is a mapping from a value set to another. It specifies how values in one value set are derived from values in another value set.
- *Semantic rule* is a text explaining the concept (usually in a natural language).

{These seem to be rather a mixture not concerning only knowledge, but also the objects identified by particular concepts, linguistic expressions representing concepts, and some relationships between concepts. Nevertheless, knowledge concerning particular concepts is probably covered by these 'primitives'. The last remark: An entity is identified by a (descriptive) *attribute*; an occurrence of a concept thus cannot identify an occurrence of another concept unless it is a concept of an attribute. For instance „the identity card number of a person" identifies a person but the number itself does not identify anything unless it is a value of the above descriptive attribute.}

The fundamental notion, *concept*, is regarded as a central epistemological unit of human knowledge. The knowledge content of a concept is its *intension.* The concepts, knowledge primitives, and the *structure* they form in the intension of a concept are called its *characteristics*. {The key question here is: What is it 'the structure they form in the intension'? Isn't it the concept itself? We would consider characteristics to be only the concepts and knowledge primitives (letting aside the peculiar character of knowledge primitives) contained in the intension of the concept. Another question: Are two concepts with the same intension one and the same concept? Maybe, that unlike Kauppi, Kangassalo intends to express here the fact observed already by Bolzano, illustrated by his well-known example [Bolzano 1837] which shows that two concepts with the same intension (content) do *not* have to be the same concepts: THE UNLEARNED SON OF A LEARNED FATHER and THE LEARNED SON OF AN UNLEARNED FATHER.} A set of objects (as well as data representing objects in the database) to which a concept *applies* is called its *extension*. {Well,

the set of objects, to which a concept applies differs dependently on possible worlds and time points. It is probably understood as the set of objects to which the concept applies in the *actual* possible world. Still this set is time-dependent. Similarly, the data corresponding to particular objects differ with respect to the database state. And, moreover, the data corresponding to the particular objects are actually given by other concepts than the objects themselves.} The elements of the extension are called *occurrences* of that concept. *A knowledge unit* is either a knowledge primitive or a concept. *Basic concept* cannot be analysed using other concepts of the same conceptual system. It can contain one or more knowledge primitives. In its most primitive form a basic concept consists of its *name* and *value set*. {A name of a concept is a linguistic expression representing the given concept. The value set concerns mostly the case of analytic concepts. Thus basic concepts can be compared with 'printable types' of other semantic data models.} Some knowledge primitives can be attached to it (e.g. semantic rules and constraints). A *derived concept* is a concept the characteristics of which have been derived from the characteristics of other concepts in the way described in the *definition* of that concept.

The basic epistemological relation between concepts is the relation of *intensional containment*. The methodology and notations used in conceptual modelling are based on this relation. {Unfortunately, this relation is not precisely defined. Kauppi [Kauppi 1967] considers it to be a primitive pre-theoretical notion. An attempt to explicate this relation can be found in [Kangassalo 1993]}: Concept A *contains intensionally* a concept B if the knowledge that forms concept A contains the knowledge that forms concept B. Note that we are talking about the knowledge required to recognise phenomena A and B in the UoD, *not how the definitions of these concepts are constructed*. {But isn't it just this definition that forms the knowledge about concept? Example given here does not reveal too much}: DOCTOR contains PERSON, DOCTOR contains SPECIAL MEDICAL EDUCATION. {But wouldn't we define a doctor as a person with a special medical education?} A rather more precise definition is provided in [Kangassalo 1998]: Concept A *contains intensionally* knowledge unit P ($A \geq P$) iff P is one of the characteristics of concept A. The relation of intensional containment (IC) is reflexive, transitive and anti-symmetric. {This claim seems to be rather strong, especially with respect to anti-symmetry. We will return to this question later after reproducing the types of IC}:

1. 'is-a',
   a driver is a person, a student is a person, etc.
   {This is a classical ISA relation; a necessary one.}
   It may be that in two concepts describing the same object there is not a single common characteristic recognised. {It may happen that particular expressions are not connected with derived concepts yet: man ≥ person ≥ living-thing ≥ entity. It is a subject of further analysis to reveal that a man, e.g., is a person with the property of being a male, and so on.}

2. 'Contains', 'has a component'
   a car has an engine, a car has tires, ...
   car ≥ engine ≥ ignition system ≥ distributor.
   {This is the 'classical' part-whole relation that is thoroughly discussed in [Junkkari 1999]. We have to distinguish whether the relation is necessary or contingent. In the latter case it is dubious to consider it to establish the IC relation.}

3. ‘is...’, ‘has’
   a person has a name, an address, an age, ..., an employee is engaged in a work-phase,...
   {This is the ‘classical’ database construct: the attribute. The question arises here: Does it have to be a necessary relation? Certainly not: A person *can* have an address but does not have to. But if it is a *contingent* relation, how can it be that a concept of person intensionally contains the concept of address? This is solved in COMIC by a supplementary condition. Moreover, if the relation of IC is enriched in this way, then it is certainly *not* anti-symmetric: A republic is represented by its president, a president is a representative of a republic. Hence REPUBLIC ≥ PRESIDENT and PRESIDENT ≥ REPUBLIC, but these are certainly not the same concepts. This situation is handled in COMIC by introducing another concept that contains both the previous concepts: A representative in this case.}

Now we come to the important notion of the whole theory, viz. the notion of definition. *A definition of the concept* is a rule or a linguistic instruction which specifies how the knowledge forming the defined concept (definiendum) is to be constructed from the knowledge given in the defining concepts and in the definition itself. In order to find out the intension of the defined concept a definition must in some cases be *evaluated*, i.e. the definiendum must actually be *constructed*. Several different definitions may all evaluate to the same concept. *Basic concept* is undefined: it cannot have other concepts as its characteristics (but it can contain knowledge primitives). Intensionally contained concepts are defining concepts, but all the defining concepts shall not necessarily be contained concepts. {Evaluating the definition in order to construct the intension of the defined concept concerns mainly analytic concepts when the rule specifying the algorithm of calculating derived values from defining values is provided. It may also be used in case of generalisation. The last claim is not in a full accordance with our intuition. The example, namely ‘penguin is a bird without feather’, that should reveal the fact that the concept of feather is used in the definition but is not contained in the concept of penguin does not reveal that. We would rather say that the concept of feather is contained in the concept of penguin: To find out whether a given bird is a penguin we have to use the concept of feather to test whether a given individual is a bird and does not have feather. (By the way, the definition is in a way improper, because penguin has special kind of feather, namely down.) Another example: „an identifying property may not be derivable from the defining concept but it introduces *new knowledge* into the defined concept“. But in another place it is affirmed that name of a person (which may serve as an identification) is intensionally contained in the concept of person.}

The following are the types of definitions that are commonly used:

1. *Aggregation*, in which a concept is defined as a collection of its characteristics (usually connected by AND).
   *Example*: PERSONNEL is a set of EMPLOYEEs. Each EMPLOYEE has a NUMBER and an AGE and an ADDRESS and a SALARY. Hence PERSONNEL ≥ EMPLOYEE, EMPLOYEE ≥ NUMBER, EMPLOYEE ≥ AGE, EMPLOYEE ≥ ADDRESS, EMPLOYEE ≥ SALARY
   {This type of definition covers a ‘classical’ (unary) attribute, as well as a part-whole relation and grouping which are usually also mapped by an attribute.}
2. *Generalisation,* in which a concept is defined as a collection of those characteristics that all its defining concepts have in common.
   *Example*: PERSON is a generalisation of an EMPLOYEE, SENIOR-CITIZEN, POLICEMAN that shares NAME, ADDRESS and AGE. Hence EMPLOYEE ≥ PERSON, SENIOR-CITIZEN ≥ PERSON, POLICEMAN ≥ PERSON.

In [Kangassalo 1993] *specialisation* is also mentioned: a concept is defined from a defining concept by specifying some characteristics which the definiendum does *not* have or by specifying some constraints on the characteristics of the defining concepts that have to be true for the definiendum.
{These are usual constructs used to define the 'ISA' relation.}

3. *Value transformation,* in which a concept is 'defined' by specifying how the values representing it can be derived from values representing the defining concepts. The intension of the definiendum is *not constructed*; it must be evaluated separately. {In this case we define derived data (definable attribute in the HIT method), i.e. a redundant attribute; the respective function by means of which values of the redundant attribute are calculated should be specified as an integrity constraint.}

*A concept structure* is a diagram which represents a *definition* of the concept. *A conceptual schema (CS)* is a definition (concept structure) of the single concept describing the whole UoD. It consists of the defined concept referring to the UoD, and of its definition hierarchy which ultimately derives the characteristics of the definiendum from the characteristics of basic concepts. Structurally it is a *directed acyclic graph* based on the relation of intensional containment. {A special role in the schema is played by *identifiers, constraints* and *conditions*. Since these roles are quite analogous to those known from 'classical' data models, we will not deal with them any more here.}

In the Section 2.3.3 *Collection of concept structures* of [Kangassalo 1993] the problem of unifying particular users knowledge is discussed. We quote here the first paragraph (with our emphases):

> Each user describes *the meaning* he or she assigns to each *concept* by giving it a concept structure in terms of more concrete or lower level concepts, until the level of *observable* or otherwise generally known concepts has been reached. In general, the level of observable concepts should be reached because there is no guarantee that 'generally known concepts' are identical for all people. One of the goals of this analysis is to find out what is really meant by a concept. For example, in one project we found out that the concept 'signature' had more than 10 different definitions used by different people. Because that concept is extremely important in some legal contexts, it was very useful that the definitions went down to the concrete level so that the differences could be clearly recognised.

{A few comments on this important passage: First, the *meaning* is assigned to expressions not to concepts. A concept is just the meaning of an expression. Hence, in our opinion, the problem with 'signature' did concern the *meaning* of this *expression* not of the concept of signature. So those ten definitions expressed *different concepts* and the problem consisted in deciding which of these concepts should be assigned to this *expression*. The approach accepted by Kangassalo is influenced by Kauppi and she might perhaps agree that the definitions „determining the intension of signature" determine different *extensions* of this concept. Yet we still feel that the problem concerned the *meaning of the expression* 'signature'. The notion of *observable concepts* is understood intuitively as those concepts that are in a way comprehensible without a definition. Unlike 'basic concepts' (which might correspond to 'printable types' of semantic data models) they may be provided with a definition in further analysis.}

Concluding this section we have to state that COMIC is a very interesting and useful tool for conceptual modelling fully exploiting Kauppi's concept theory. There is also an interesting attempt to explicate the relation of intensional containment which is a primitive

notion for Kauppi. Though there are many problems connected with it (which have been formulated above), its seemingly simple idea — using *only* the relation of IC as a basic modelling construct — which might seem to be insufficient in practice is balanced by using different types of 'definitions' and by the fact that conceptual schema is in fact a *definition* of the UoD. Thus the traditional modelling constructs, such as a generalisation, specialisation, aggregation and attribute are as if 'hidden' from the user, which may be an asset of this approach. Yet some questions (that will be answered mainly in Section 3.3) remain.

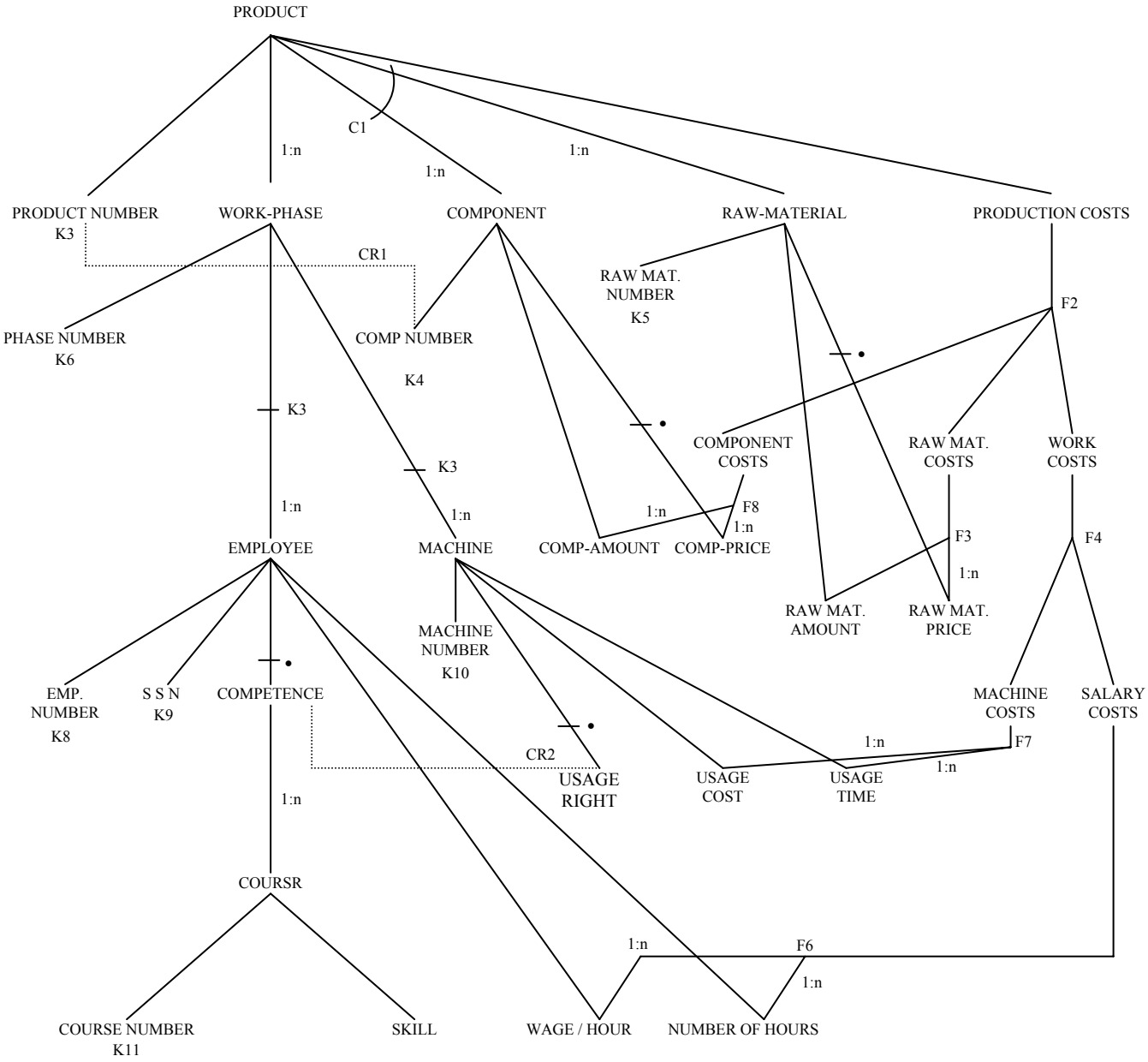
First, we miss the distinction between empirical and analytical notions. Thus the definition of an *extension* of a concept seems to be peculiar. If it is an empirical concept, then the extension of this concept, i.e. "a set of objects (as well as data representing objects in the database) to which a concept *applies*" is probably meant to be the set of objects in the actual possible world, but there are many such sets in different time points. The data representing the objects in the database are something completely different from the objects themselves and they can hardly be considered to belong to the same set, the extension. Moreover, there is no means to distinguish between an 'empirical function' (such as a 'name of a given person') and an 'analytical function' which is used in the 'definition' by value transformation. There is also no means to distinguish between traditional 'entity sets' and 'value sets' which is considered to be an advantage of the approach.

Second, it is not clear, how to model such complex concepts as the concepts of $n$-ary empirical functions ($n \geq 2$) without introducing some equivalent of the concept of Chen's relationship set, which may be a stumbling block of the analysis. Consider, for instance, the concept of HIT-attribute expressed by 'periods (determined by the day of a week and an hour) when a given subject is lectured by a given professor in a given room for a given group of students' of the 'type'
(SUBJECT, PROFESSOR, ROOM, STUDENT-GROUP) $\rightarrow$ ((DAY, HOUR) $\rightarrow$ BOOL). Which concepts are intensionally contained in which concept here? Without introducing the concept of LECTURING which in fact corresponds to the concept of relationship set we can hardly determine particular IC relations. But then we have only the possibility to map unary empirical functions at our disposal and the user is forced to recognise the concepts of relationship sets at the very beginning of the design work. But it is just this feature which is promised to be avoided by this methodology.

Third, there are some special 'descriptive' concepts like ADDRESS, DATE, and so on, which are 'contained in' many other concepts. According to this methodology we have to actually introduce new concepts like the ADDRESS OF A PERSON, the ADDRESS OF AN ORGANIZATION, ... But how to solve then the situation when the user finds out that he/she would often like to ask questions like 'what is located on a given address'? ADDRESS thus becomes an independent object of interest ('similar' to our entity set, 'marked' descriptive type [Duží 1999]) with its own identifier, but it is 'hidden under' other concepts in which those particular concepts of address are contained. In this case we have to introduce a concept of ADDRESS into the schema, and the other concepts of address will be contained in it, which is at the best peculiar.

Fourth, it is not clear how to model the situation when there are two different kinds of IC relation between two concepts. For instance, in the definition of the concept PRODUCT ([Kangassalo 1993] Fig. 9, p. 299) aggregation is used to define product. Well, according to this definition a product is assembled from components and raw materials, hence a component is contained in product and a raw material is contained in product. But a component is a product as well, hence there is another IC relation between COMPONENT and PRODUCT, which is not recorded here. Moreover, the distinction between a raw material and a

component is thus not recorded as well. Actually, these two objects have exactly the same attributes (contain the same concepts). Why then using two different concepts for them?




Last but not least, there is an essential objection to this conception: Contingent relations should not be considered to establish the IC relation. Affirming, e.g., that the concept of a person intensionally contains the concept of a name, address, identity card number, etc. (person ≥ name, person ≥ address, person ≥ identity card number), we actually claim that it does 'follow from the concept' of a person the he/she has a name, address, identity card number, etc., which is certainly *not* true. A person *can* have a name, address, identity card number, but does not *have* to. This is solved in COMIC by assuming that the expression 'person' is connected with such a concept of a person that is suitable from the 'IS point of view'. The fact that the above relation may be a contingent relation is recorded as a condition.

### 3.1.2. Related works of Finnish researches

In this section we briefly recapitulate the main results of some of the Kangassalo's followers, namely M. Junkkari, T. Niemi, J. Palomäki, M. Niinimäki and J.F. Nilsson. A common feature of these works can be characterised as exploiting Kauppi's concept theory for various functional and algebraic representations of concept relationships or operations that associate concepts.

In [Niinimäki 1999] an algebraic approach to Kauppi's concept theory is applied. A large part of Kauppi's theory is described and an abstract implementation of it is formed. Some extensions of the concept theory are presented, thus obtaining, e.g., the result that a set of traced concepts instead of a unique concept can be returned as a value of the specified function. Thus the structure of the concept system proposed does not fully comply with the strict axiomatisation of the original theory. By means of the relation of the intensional containment four association relations between two concepts are defined. They are: 'Homogen-compatibility' (comparable, compatible), 'Heterogen-compatibility' (Incomparable, compatible), 'Opposition' (comparable, incomparable) and 'Isolation' (Incomparable, incompatible). The paper presents interesting theoretical results, unfortunately without any reference to the semantics of the intensional containment (IC) relation. This relation is neither defined, nor even intuitively explicated and none reasonable examples are presented. Nevertheless, the authors claim that „It is suggested that intensional containment relation covers the ISA relation, aggregation and the component relation. Whatever view is taken towards the semantics of the IC, the representation we employ here gives powerful tools for analysing different kinds of knowledge structures". It need not be true in case of the aggregation, because in that case the IC relation does not have to be a partial ordering.

Nilsson and Palomäki discuss in [Nilsson 1998] handling intensional aspects of concepts with computational concerns. Concept net is here based on *binary* relation between concepts, namely the 'is-a' relation. The concept net formalism is reformulated into relational algebra which is next embedded in predicate logic and to the possible extent even in definite clauses of logic programming. Quoting the basic presumptions:

> "The fundamental concept relationship is the ISA relationship, giving rise to a distinction between extension and intension. The intension and extension of a concept can be defined as follows:
> *The intension* of a concept is the information content inherent in the concept, which enables us to recognise the individuals falling under the concept.
> *The extension* of a concept is the set of all those individuals which fall under the concept. An individual falls under the concept if the concept applies to it."

Ignoring the category of possible worlds, the authors have no means to distinguish between a *contingent* and *necessary* 'is-a' relation. Yet a concept intensionally contains another concept only in case of a necessary 'is-a' relation, i.e. the ISA relation. Thus, e.g., necessarily each student is a person, i.e., in all possible worlds and time points (the concept of a student contains the concept of a person), whereas the fact that Plato is a philosopher is a contingent fact: It is possible that Plato were something else and there are time points (even in the actual possible world) when he was not a philosopher, for instance in his childhood. Thus the concept of Plato does *not* contain the concept of a philosopher. Moreover, the extension is not well defined, because there is not *one* set of individuals which fall under the given concept but infinitely many of them dependently on possible worlds and time points. Thus claiming that the extension is determined *uniquely* by the intension of a concept is not correct. Irrespective of the above flaws the paper brings very promising results for computer

scientists. Building a concept complete distributive lattice first (with its rather peculiar universal (upper) and bottom concepts), via algebraisation of concept nets and embedding it in predicate logic, the authors propose a concept logic which could be probably easily understood as a logic programming tool.

T. Niemi in [Niemi 1998] presents a good survey of Kauppi's concept theory. He presents definitions of intensional product and sum, of comparable and compatible concepts, intensional negation of a concept, and intensional difference and quotient of concepts. Intensional approach in the information modelling area is stressed in contradistinction to the extensional approach. The author correctly states that the extensional approach is insufficient, for if extensionality is strictly followed, an extensionally defined concept (as a set of individuals) changes every time a new instance is added. Another problem is that two different concepts can have the same extension. Following Kangassalo, he defines the intension of a concept as the information content of the concept, i.e. all information contained in it, while the extension of a concept is the set of all individuals actualising the definition of the concept (this extension can change according to possible worlds and time points). The main contribution of the paper consists in algebraic presentation of the complete distributive concept lattice. The fundamental relation between concepts is, of course, the intensional containment relation (IC). All the results presented here are valuable, but unfortunately valid only when the IC relation is taken to be the ISA relation. The author's ambitions fail in his effort to enrich the IC relation (in accordance with Kangassalo) to cover also the 'part-whole' relation and 'being an attribute of ... ' relation. As we have stated above, in case of 'being an attribute of ...' the relation is not anti-symmetric (hence it cannot be considered to be a partial ordering), and in case of the 'part-whole' and 'being an attribute of ...' relations the inverse inclusion relation between the extensions of concepts does *not* hold. Thus it is certainly not true that the set of universities is a subset of the set of professors. Similarly, it is certainly not true that the set of cars is a subset of the set of motors (as author claims). The trivial error consists here in not distinguishing the property of 'being a motor' from the property 'having a motor' (as a proper part). Thus the set of cars is (in all the states-of-affairs) a subset of all the individuals that have a motor (as a proper part), similarly a set of motors is a subset of the set of all the individuals that have a motor (as an improper part), but the above claim is obviously *not* valid. The inverse inclusion relation between the extensions of concepts and the contents of concepts holds only in case of the conjunctive composing particular components of the concept (as it was shown already by B. Bolzano [Bolzano 1837], see [Palomäki 1997]). Well, Kauppi claimed that this relation holds, but (as J. Palomäki told me in a private discussion) she admitted that the best interpretation of the IC relation is the ISA relation and she never offered any other.

### *3.2. HIT data model*

The Czech school is represented by the HIT data model [Duží 1986, 1992, 1997, 1999], [Zlatuška 1986, 1990]. HIT is an acronym for Homogeneous, Integrated, Type-oriented. It is essentially an object-function model [Scholl 1990], the specification and manipulation tool of which is not an object algebra but the 'language of constructions' that can be viewed as a modified version of the transparently understood typed λ-calculus with tuples. HIT *conceptual schema* is defined as a couple $<A_S, C_A>$, where $A_S$ is the set of concepts (constructions) of HIT-attributes over a base of 'sorts' S, and $C_A$ is the set of constructions of consistency constraints connected with attributes of $A_S$. From the theoretical point of view, HIT data model is based on the Transparent Intensional Logic (TIL), see [Tichý 1988], and to explicate these notions precisely, we are going to briefly summarise basic principles of TIL.

### 3.2.1. Transparent Intensional Logic.

#### (a) Type system.

The basic modelling construct of our apparatus is not a relation but a *function*. This choice is sufficiently motivated in [Materna 1998] (functional approach with its operations of application and abstraction enables us to create non set-theoretical complexes that are not realisable in the relational approach) and there is not much to add to it. Perhaps just one remark: Functions and relations are, in a way, equivalent when not considering partiality; but our functions can be partial, which enables us to catch even contingent relations between concepts. We need to work with functions as with 'full-right' objects, which is not possible within the first order predicate theories. Therefore we use the type system in which we need to have the possibility of giving functions as arguments of other functions, functions that give other functions as a result, etc. Hence the classical ADT approach is not sufficient. A slightly modified version [Zlatuška 1986] of the simple theory of types [Church 1940] meets this demand. (TIL is nowadays based on the ramified theory of types, for when analysing natural language expressions, we need to work even with concepts (constructions) as with 'full-right' objects, which is not possible within the simple theory of types. Nevertheless, for the purposes of conceptual modelling the simple theory of types will do.)

The existence of a *base* is assumed. The base is a collection of mutually disjoint non-empty sets which are elementary types. Over these elementary types an infinite hierarchy of types is built by means of functional types. Beside the functional type we use a tuple type as well. Although a tuple object might be created by means of a function, using the tuple construct makes the system easier to use.

**Definition 1**. Let $B$ be a base.

i) Every member of the base $B$ is an (*elementary*) *type* over $B$.

ii) If $T_1$, $T_2$ are types, then $(T_1 \rightarrow T_2)$ is a (*functional*) *type*, i.e. the set of all (partial) functions from $T_1$ into $T_2$.

iii) If $T_1,...,T_n$ ($n > 1$) are types, then $(T_1,...,T_n)$ is a (*tuple*) *type*, i.e. the Cartesian product of $T_1,...,T_n$.

An element O of a type $T$ is called an *object of the type T* (or a $T$-object), and denoted O/$T$. □

#### (b) Intensions & extensions.

Another weakness of generally used predicate logics (even of a higher-order) is the fact that they usually do not distinguish between intensions and extensions. This distinction plays an important role in the theory of semantic information connected with data (see Ch. 4). Actually, when we want to logically define (informationally) redundant data structures, to compare data as for their informational capability, etc., we cannot do that without an apparatus enabling us to work with intensions and extensions.

In the database literature intensions and extensions are often understood in the following sense: An intension is a rule specifying a function or a relation, whereas an extension is a table of actual values of the function or relation [Bell 1988]. In TIL (as well as in other logics based on possible-world semantics) an intension is an abstract object — a function — the domain of which is the so-called logical space, i.e. the set of possible worlds (logically consistent states-of-affairs relative to a given language). These two approaches have something in common. Imagine, e.g., an expression 'Age (of a given person)'. The entity denoted by this expression shows a quasi-functional behaviour: given a person, you can

associate this person with just one value of his/her age. But the age of a person is time-dependent, and moreover, knowing the notion of age does not suffice for computing the age of a person. We have to investigate it, to examine the actual state-of-affairs to find out what the actual age of the person is. This is due to the fact that the entity denoted by the expression 'Age of ...' is an intension (in the TIL-sense). In fact, this is the reason why we need to store data to a database; we have to pick them up in the reality, for they are values of intensional functions in the actual states-of-affairs, and the above mentioned 'rules specifying functions or relations' usually identify (construct) TIL-intensions. But since such rules could also specify „normal" extensional objects, like e.g., 'average', 'sum', etc., we shall always understand intensions and extensions in the TIL-sense. Another use of the words "intension", "extension" has been introduced in Section 3.1.1: Intension/extension *of a concept.* These notions will be precisely defined in Section 3.3. Referring for details to [Tichý 1988] we recapitulate:

The base in TIL is called an *epistemic base*, and it is a collection of four elementary types {o, ι, τ, ω}:

- **type o** is the set of truth values {True, False},
- **type ι** is the universe of discourse and its members are individuals (ι is really 'universal', there is only one universe of discourse, the same for all possible worlds; there are no 'possible individuals'. Moreover, an individual is understood as a 'bare' entity without any non-trivial essential properties),
- **type τ** is the set of time points or real numbers playing also the role of their surrogates,
- **type ω** is the set of possible worlds. Intuitively, a possible world is a logically possible state-of-affairs. To explicate this notion more precisely, we need some preliminaries: First of all, there is a collection of intuitively, pre-theoretically given traits assigned to objects of our interest. The choice of the universe of discourse and of the basic traits depends, of course, on the area we want to investigate. In case of objects and traits being empirical, the distribution of the traits among the objects is unpredictable and we have to apply an empirical procedure to find out which of the logically possible distributions is the actual one. But these possible distributions can change in time: Hence *possible world* is defined as the chronology of logically possible distributions of basic traits among objects (*facts*), and the logical space of a language is the set of all possible worlds.

Over the epistemic base intensions and extensions are inductively defined as follows:

**Definition 2.**

i) A $T$-object, where $T \neq (\omega \rightarrow T')$ for any $T'$ , is an *intension of the 0th order* or an *extension.*

ii) Let a $T$-object be an intension of the $n^{\text{th}}$ order. Then $(\omega \rightarrow T)$-object is an *intension of the $(n+1)^{st}$ order*.

iii) Only what satisfies i), ii) is an *intension.*

Further we shall use the term *intension* only for intensions of the $n^{\text{th}}$ order, where $n > 0$. □

Natural language expressions usually denote objects (intensions) of a type $(\omega\rightarrow(\tau\rightarrow T))$, for some type $T$. Whenever we shall not need to work separately with parameters of the types ω, τ, time dependent intensions will be handled as if they were of a type $((\omega,\tau)\rightarrow T)$, which will be abbreviated by $(\omega\tau\rightarrow T)$. The couple (ω,τ) will be called the *state-of-affairs*.

Sets (classes) are in our approach modelled by their characteristic functions. Hence a class of $T$-objects is an object of the type $(T \to o)$, i.e. a function which returns the value True for each object belonging to the class, and False for all the other objects, as it is obvious from the following example:

*Examples* of intensions and extensions

| extension | intension |
|---|---|
| class of $T$-objects<br>$(T \to o)$ | property of a $T$-object<br>$(\omega\tau \to (T \to o))$ |
| n-ary relation-in-extension<br>$((T_1,...,T_n) \to o)$ | n-ary relation-in-intension<br>$(\omega\tau \to ((T_1,...,T_n) \to o))$ |
| {True, False}<br>o | proposition<br>$(\omega\tau \to o)$ |
| analytical function<br>$(T_1 \to T_2)$ | empirical function<br>$(\omega\tau \to (T_1 \to T_2))$ |

**(c) Constructions**

Ways in which objects are obtained are called *constructions* of objects and they can be intuitively understood as transparently viewed terms of the typed lambda calculus [Barendregt 1981], [Zlatuška 1993] which are interpreted in the fixed way. Philosophically, they can be characterised as abstract procedures, consisting of some intellectual steps that lead to an identification of the constructed object (or in some well defined cases they fail to identify anything; this is the case of strictly empty concepts like the greatest number that represents an improper construction, see below). Hence constructions consist of parts, but the *way of composing* these parts together is important, not only the parts themselves, thus forming *complexes* that are not reducible to set-theoretical entities (unlike, e.g., Cresswell's tuples [Cresswell 1985]). They are inductively defined as follows: Assuming that for any type there are an unlimited number of representatives, variables, and that there is a total function called valuation that assigns one object of the given respective type to each variable, we define:

**Definition 3.**

i) Atomic constructions are *variables*. A $T$-variable $x$ $v$-constructs a $T$-object which the valuation $v$ assigns to $x$.

ii) If X is an object whatsoever, then $^0$X is a construction called *trivialization.* $^0$X constructs simply X without any change.

iii) Let $F$ be a $((T_1,...,T_n) \to T)$-construction, $A_1,...,A_n$ be $T_1$-,...,$T_n$-constructions ($n > 0$), respectively. Then $[F\ (A_1,...,A_n)]$ is a $T$-construction called *application* (*composition*) of $F$ to $(A_1,...,A_n)$. If $F$, $A_1,...,A_n$ $v$-construct objects F, $A_1$,...,$A_n$, respectively, and if the function

F is defined on $A_1,...,A_n$, then $[F\ (A_1,...,A_n)]$ *v*-constructs the value of F on the tuple $(A_1,...,A_n)$. Otherwise, $[F\ (A_1,...,A_n)]$ is *v*-improper, i.e. it does not *v*-construct anything.

iv) Let $x_1,...,x_n$ be pairwise different $T_1$-,...,$T_n$- variables and *A* a *T*-construction. Then $\lambda x_1...x_n A$ is a $((T_1,...,T_n) \rightarrow T)$-construction, called λ-*abstraction* (*closure*) of *A* on $x_1,...,x_n$. It *v*-constructs the following function F: Let $A_1,...,A_n$ be $T_1$-,...,$T_n$-objects, respectively, and let *v'* differ from *v* only by assigning $A_1,...,A_n$ to $x_1,...,x_n$, respectively. Then F gives on $A_1,...,A_n$ as its value the object *v*'-constructed by *A* if *A* is not *v*'-improper, otherwise F is undefined on $A_1,...,A_n$.

v) Let $A_1,...,A_n$ be $T_1$-,...,$T_n$-constructions, respectively. Then $(A_1,...,A_n)$ is a $(T_1,...,T_n)$-construction called *tuple*. If $A_1,...,A_n$ *v*-construct objects $A_1,...,A_n$, respectively, then $(A_1,...,A_n)$ *v*-constructs an object $(A_1,...,A_n)$ of the tuple type $(T_1,...,T_n)$, otherwise it *v*-constructs nothing, it is *v*-improper.

vi) Let *A* be a $(T_1,...,T_n)$-construction, then $A_{(1)},...,A_{(n)}$ are $T_1$-,...,$T_n$-constructions, respectively, called *projections*. If *A* *v*-constructs an object $(A_1,...,A_n)$, then $A_{(1)},...,A_{(n)}$ *v*-construct objects $A_1,...,A_n$, respectively.

vii)Nothing is a construction but a *T*-construction for a type *T* due to i) - vi). □

Variables are *incomplete constructions* that construct dependently on valuations. This is an objectual version of the Tarskian conception of variables. There is however an essential distinction: Whereas not only Tarski but nearly every standard logician consider variables to be letters, characters, this conception is untenable in TIL; constructions are *language independent* entities. Hence the letters standardly used for variables like *x, y, z, ...* are *names of variables* here.

Trivialization ($^0$) is a special construction that might seem to be dispensable. It is an immediate, simplest way of constructing an object. Nevertheless, it is a very important construction, enabling us, among others, to distinguish between 'using' and 'mentioning' concepts [Duží 1994] and to distinguish between a concept of an object and the object itself.

A closed construction, i.e. a construction without any free variables (where variables can be either λ-bound — when being in the scope of a λ-operator and not within a trivialisation, or o-bound — within the scope of trivialisation) meets all the intuitive demands stated for the meaning of a natural language expression: It is an objective, non-linguistic structured entity that constructs (identifies) an object. Hence we conceive *meanings* of expressions, i.e. *concepts,* as *closed constructions* [Materna 1998]. (For the sake of simplicity we identify here particular closed constructions with concepts; they are, in fact, concepts*. For details, see [Materna 1998] where concepts are defined as *classes* of quasi-identical constructions, namely classes of such constructions that are indiscernible from the conceptual point of view; in particular, in a natural language we cannot express two quasi-identical constructions by two different expressions.) See, however, Section 3.3 for details.

A *T*-construction, i.e. a construction constructing an object of the type *T*, will be rather non-precisely called a construction of the type *T*.

Logical connectives, less than and equal tests, quantifiers, etc., are (analytical) functions, i.e. objects of the respective (functional) types. Logical connectives and tests need not be defined here. Quantifiers are defined as follows:

Let *x* be a variable of a type *T*, *B* a construction of the type o.

$\Pi^T/((T \to o) \to o)$ are *general quantifiers*; $[^0\Pi^T\ \lambda x\ B]$ constructs True if $\lambda x\ B$ constructs the class of all members of $T$, otherwise False.

$\Sigma^T/((T \to o) \to o)$ are *existential quantifiers*; $[^0\Sigma^T\ \lambda x\ B]$ constructs True if $\lambda x\ B$ constructs a non-empty class, otherwise False.

$I^T/((T \to o) \to T)$ are *singularizers*; If $\lambda x\ B$ constructs a $T$-singleton K, then $[^0I^T\ \lambda x\ B]$ constructs the only member of K, otherwise $I^T$ is undefined on K.

*Notational conventions and abbreviations*:

- Negation $[^0\neg\ A]$ will be abbreviated by $\neg A$.
- When applying logical connectives, identities, tests, etc. we use infix notation without trivialization.
- Instead of $[^0\Pi^T\ \lambda x\ B]$ we write $\forall x\ B$
- Instead of $[^0\Sigma^T\ \lambda x\ B]$ we write $\exists x\ B$
- Instead of $[^0I^T\ \lambda x\ B]$ we write $\iota x\ B$ (the only $x$ such that $B$)
- We will standardly use variables *w, t* as ranging over $\omega$, $\tau$, respectively. When applying an intension A of a type $(\omega\tau) \to \alpha$, for some type $\alpha$, at $w$, $t$, we will use an abbreviation $^0A_{wt}$ instead of $[[^0A\ w]t]$.

*Note*: We shall use the same sign 'ι' to denote the type of individuals as well as the singularizers, since no confusion can arise.

### 3.2.2. Base of sorts

In data modelling the role of elementary types is played by *sorts*. This is due to practical reasons. When creating a database schema we need to determine 'nodes' of the schema which stand for some basic classes (sorts) of individuals. Speaking about objects of our interest we do not use formulations like „the class of individuals which have the property of being an employee". Instead we simply speak about an employee, student, material, etc. These expressions denote various properties (not necessarily of individuals only), and we demand that the information system in question should offer information on any object which has the selected property. This means among others that the attributes of a schema should concern — and thus be restricted to — only objects with this property.

Nearly all the current data models use two kinds of types: *entity sorts* and *descriptive sorts*. (An entity sort is essentially the same as an 'abstract' type, a descriptive sort corresponds to a 'printable' type of [Hull 1987].) But as we have seen above, these two categories are usually defined vaguely, entities as being identifiable objects, things of the real world, and descriptions as values, strings, etc. Exact explanation can be given using TIL. Referring for details to [Materna 1987], we briefly recapitulate.

An *entity sort P* is given solely by a property (of objects of a type $\alpha$, i.e. an $(\omega\tau\to(\alpha\to o)$ − object), say P. It is the union of classes which are selected by this property through a relevant time interval, i.e. the set of all objects which possessed, possess and will possess the property P. Let S be a time span relevant for the purpose of the given information system. Let $C_i$ be classes of objects selected by the property P during S, i.e., each $C_i$ is the value of P in the actual possible world and a time point $s_i \in S$. *Entity sort P* is then the union $\cup_i C_i$. Classes $C_i$ will be further called *populations* of the sort *P*.

The fact that an entity sort is given solely by a property has an important consequence: It is not a recursive set and it is not representable ('printable') in the following sense: A class C is *representable* iff there is a recursive injection of C into a set A*, where A* is the set of

all finite strings over a finite alphabet A. If an entity sort *P* were representable, we would have to be able to use a recursive function taking the property P as its argument and returning the class generated by P in the actual world. But since we can never know which of the possible worlds is the actual one, no such function can be found. A *descriptive sort* is then any recursive, i.e. representable class.

The *base in the* HIT *data model* is a collection of entity sorts *E*, descriptive sorts *D*, time points τ and possible worlds ω: {*E, D,* τ,ω}. The set of truth values o is considered to be a descriptive sort. There are some problems connected with this conception of the type system, namely the base of sorts. We will return to it in the Section 3.3.

Members of descriptive sorts are encoded by data stored in a database by means of which we describe entity sorts, i.e. they are ranges of extensional (analytical) functions defined by attributes (empirical functions) in particular states-of-affairs. (By a rather non-precise term 'extensional function' we denote an analytical function, i.e. a (functional) object that is the value of an attribute in a state-of-affairs.) The impossibility of finding a recursive function determining the population of an entity sort, as well as the intensional character of attributes (which will be discussed in the next chapter) have thus to be compensated by cognitive actions (experience, data collection).

### 3.2.3. HIT attributes

Traditionally, philosophy and logic preserved the term 'attribute' for properties (and, as the case may be, for relations, as properties of tuples). The development of databases and data models in seventies was connected with a broader interpretation of the term 'attribute': 'The address of a person', 'the salary of an employee', 'the children of a man and woman', are examples of attributes in this broader sense. Indeed, they are not properties (nor relations); though it is possible to predicate of a person that he/she has the address of him/herself, we are interested in the value of the address. Hence whereas a property selects in every possible world a class (i.e. an object of a type $T \rightarrow o$), an attribute (in this broader sense) selects an (extensional, analytical) function (i.e. an object of a type $T_1 \rightarrow T_2$). For instance 'the address of a person' selects a function which associates each person with its address. That attributes are *intensions* (empirical functions) is obvious: A person can move, i.e., the address of a person changes in time, and it is not logically necessary that the person has a given address, say A. He/she could live anywhere else.

Attributes determined by concepts of a HIT conceptual schema are restricted to empirical functions of the so-called *simple types*. (This restriction is valid only for the basic conceptual schema. Attributes of derived schemata — views — are of more complicated types [Zlatuška 1986]. Anyway, this restriction will not influence the degree of generality of our considerations.) There are two classes of simple types:

a) $(\omega\tau \rightarrow (T_1 \rightarrow T_2))$
b) $(\omega\tau \rightarrow (T_1 \rightarrow (T_2 \rightarrow o)))$

where $T_1$, $T_2$ are types of sorts or tuples of sorts. Hence an attribute selects in every state-of-affairs a function mapping a sort or a tuple of sorts to a sort or a tuple of sorts (case a)), or to their power set (case b)). We will call attributes of the type a) *singular attributes*, and attributes of the type b) *multivalued attributes*. A singular attribute may also be a property (in this case $T_2 = o$); $T_1$, and $T_2$ in case of a multivalued attribute are not equal to the sort o.

Note that our notion of attribute is a broader one than traditionally used. It covers a *tuple* construct as well as a *set* construct, and, moreover, it also covers *relationships* between

entity sorts. Thus, similarly as in ORM, we model descriptive attributes and relationships (associations) between entity sorts in a unique way, which makes the model extremely stable (see, however, Section 3.2.5). *Attribute names* are standardised natural-like language expressions. Though their exact syntax will be described in Section 3.2.5, we now use them to illustrate how the names are easy to read.

*Example*: MATERIAL, SUPPLIER entity sorts, NAME, MONTH, QUANTITY descriptive sorts:

$A_1$: '(NAME) of a (#MATERIAL)' / 1,1:0,N
of the type ($\omega\tau \rightarrow$ (MATERIAL $\rightarrow$ NAME))
singular ('descriptive') attribute

$A_2$: '(#SUPPLIER)-s who deliver a (#MATERIAL)' / 0,M:0,N
of the type ($\omega\tau \rightarrow$ (MATERIAL $\rightarrow$ (SUPPLIER $\rightarrow$ o)))
multivalued attribute ('nested relation', relationship)

$A_3$: 'Accepted (QUANTITY) of a (#MATERIAL) from a (#SUPPLIER) per a (MONTH)' / 0,1:0,N
of the type ($\omega\tau \rightarrow$ (MATERIAL, SUPPLIER, MONTH) $\rightarrow$ QUANTITY))
singular attribute with tuple ('relationship')

Using just one modelling construct — attribute — enables us to formalise and exactly handle all the operations on general data structures, i.e. derived data, views, integrity constraints, ISA relationship as well as multiple inheritance (with a semilattice of types) [Zlatuška 1986], [Duží 1986]. In this work we show that it is possible to formally compare informational capability of different data structures, e.g., of a flat relation and a nested relation, etc. (see Section 4).

### 3.2.4. Consistency constraints

Consistency constraints are propositions that specify admissible states-of-affairs (and consequently admissible database states). There are some states-of-affairs that are 'a priori' excluded. For instance, such a possible world in which the age of a person would decrease is (analytically) impossible. But we have also to exclude such states-of-affairs that are logically possible but that contradict some empirical laws or conventions of a given organisation. Consistency constraints excluding the former are called *analytical constraints* (the term 'intrinsic' is also used in some models), constraints excluding the latter are called *empirical constraints* ('extrinsic'). This distinction is significant for studying dynamic and static constraints [Vianu 1987], since empirical constraints can evolve in time, which may have significant consequences for further design and implementation phases. Hence consistency constraints help us to check the correctness of the attribute values (the correctness of data collection) and consequently the correctness of a database state. The problem is that in the majority of data models consistency constraints are formulated only in a natural language, or, as the case may be, directly in some programming language, so that their exact conceptual specification is neglected. Of course, when consulting the business reality with the user, the designer has to use a natural language. But since consistency constraints are the second constituent of the HIT conceptual schema, the task of a HIT-designer consists in analysing these natural language assertions, i.e. in transforming them into the respective logical constructions.

*Example*:

a) Analytical constraint 'the age of a person never decreases' connected with attribute

A: ‘(AGE) of a (#PERSON)’ is recorded by the following construction:

$\forall w\ t_1\ t_2\ pers\ ((t_1 \geq t_2) \supset ([^0A_{wt1}\ pers] \geq [^0A_{wt2}\ pers]))$;

(variables $w$, $t_1$, $t_2$ ranging over possible worlds and time points, respectively, *pers* ranging over the sort PERSON.)

b) Empirical constraint ‘For each material there is always a supplier’ connected with attribute

B: ‘(#SUPPLIER)-s of a (#MATERIAL)’ is recorded by the following construction:

$\lambda w\ \lambda t\ \forall mat\ \exists sup\ [[\ ^0B_{wt}\ mat]\ sup]$

(variables $w$, $t$ ranging over possible worlds and time point, respectively, *mat* over the sort MATERIAL, *sup* over the sort SUPPLIER).

*Note* the form of the respective constructions: $\forall wt$ … concerns analytical, $\lambda w \lambda t$ … empirical.

### 3.2.5. The techniques of building up the HIT conceptual schema.

Having illustrates basic principles and HIT philosophy, we now define HIT conceptual schema:

**Definition 4.** HIT *conceptual schema* is a couple $<A_S, C_A>$, where $A_S$ is the set of concepts (constructions) of HIT-attributes over a base of sorts S, and $C_A$ is the set of constructions of consistency constraints connected with attributes of $A_S$.

When building up the HIT conceptual schema, we have in mind one of the most important principles, viz. the *user involvement* in the analysis. Being confronted with the user’s (expert’s) utterances, the designer has to transform these utterances, if they are meant as proposals of attribute names, to the respective constructions (concepts) of attributes. Let us follow with some very simple examples:

*Example*:

(User:) We wish to follow “Day, month and year of the birth of an employee”

‘Day’, ‘month’, ‘year’ are evidently names of sorts (in this phase it is not important whether these sorts are descriptive or entity ones) members of which will be returned by the attribute when it is applied to a member of the sort ‘employee’. Calling this attribute ‘Day, month, year making up the date of birth of an employee’, we can write down the respective construction:

$\lambda x \quad \iota\ (y, z, u) \quad ([\text{Date-of-birth}\ x] = (y, z, u))$,

argument type — value type — attribute ‘body’

where $x, y, z, u$ are of the types EMPLOYEE, DAY, MONTH, YEAR, respectively. (We will return to the problem of ‘types’ later in Section 3.3.) This simple example reveals also the usefulness of the tuple type and tuple construction (which are not introduced in the ‘standard’ TIL) in data modelling: We need functions that return tuples as their values, and using singulariser in this example would not be correct without tuples.

Similarly ‘Children of a given man and woman’ is the name of the attribute constructed by

$\lambda x\ y\ \lambda z\ [[\text{Children-of}\ (x, y)]\ z]$

where *x, y, z* are of the types MAN, WOMAN, PERSON, respectively.

Note that in case of a singular attribute there is a singulariser specifying attribute value and identity in the attribute body, whereas in case of a multivalued attribute there is a lambda operator and application, respectively.

Theoretically, the designer has to distinguish three components in the expert's proposal:

i) the sorts making up the value type

ii) the sorts making up the types of argument

iii) the attribute „body"

The designer gradually transforms the expert's proposals into the 'language of constructions', following the points i) – iii). It is, however, very important to optimise co-operation between the designer and the expert (user). Now, verifying whether what is recorded as a construction is really that what is meant by the expert, the designer cannot discuss this by offering the expert the respective construction; the expert is not bound to know the formal apparatus ('the language of constructions') used by the HIT methodology. Instead, the designer uses a graphical support together with a standardised natural-language like names of attributes, which proved to be very intelligible for the users. In practice, the designer's method consists in confronting the user with the respective graphs supplied with natural-language like standardised names. The designer need not first write down the constructions and then transform them into graphs. The correspondence between the constructions and graphs (and natural language names) is unambiguous and it is given by the following *correspondence rules*:

R1) *Sorts* are represented by (small) circles; the circles representing the entity sorts have, moreover, a cross, in the following way: ⊗; (In the first phase, this distinction is not important and need not be represented except in the obvious cases.)

R2) Let

$A_1$ be an attribute of the type $\omega\tau \to ((T_1,...,T_m) \to S_1)$

$A_2$ be an attribute of the type $\omega\tau \to ((T_1,...,T_m) \to (S_1,...,S_n))$

$A_3$ be an attribute of the type $\omega\tau \to ((T_1,...,T_m) \to (S_1 \to o))$

$A_4$ be an attribute of the type $\omega\tau \to ((T_1,...,T_m) \to ((S_1,...,S_n) \to o))$.

The (oriented) graphs representing $A_1$, $A_2$, $A_3$, $A_4$ have the forms as defined and illustrated by Fig. 1. The graphs of the forms $(A_1) - (A_4)$ are called ***A-schemas***.

*Gloss*: The upper circles represent the sorts $T_1, ...,T_m$ and the bottom circles represent the sorts $S_1, ..., S_n$.

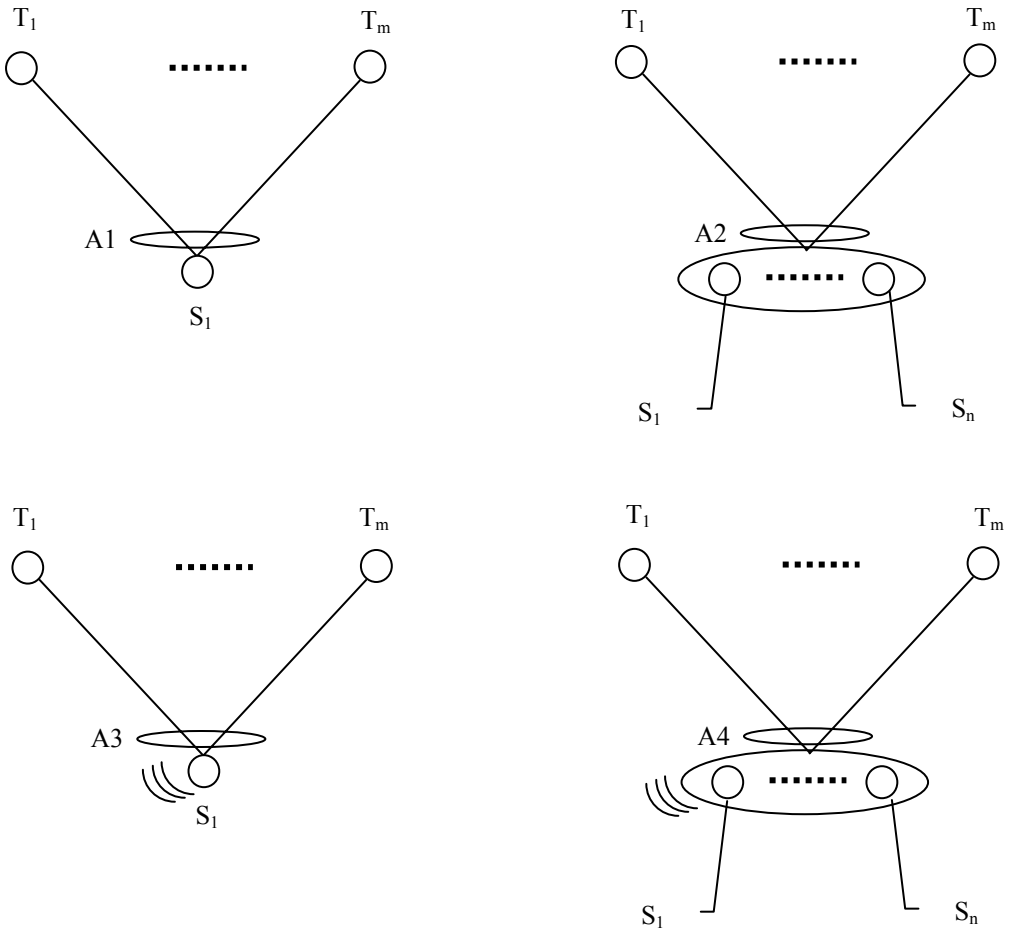
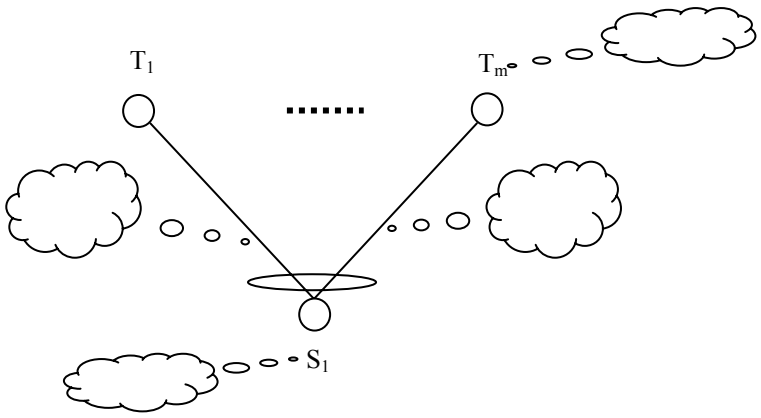



**Figure 1**: *A-scnemas*

R3) A natural standardised *attribute name* containing the names of components i) – iii) is chosen and its particular parts are written down, beside, above or below the parts of the A-schema as follows:



The names of sorts are written to the respective circles; the edges connecting the upper and lower part of the A-schema are provided with expressions (included in 'bubbles') like 'of...', 'in...', 'who works on...', etc. which together with the names of the sorts constituting the value type and argument type make up the attribute name formulated in natural language, expressing the concept, i.e. the semantics of the attribute. A 'bubble' can be ascribed also to the value type and to the n-th argument. Tuple types serving as a range of an attribute are also provided with names. The name of attribute is read 'bottom up', and it has to be a grammatically correct natural language expression, which serves as a check of the correctness of the name.

The names of attributes can also be recorded in a standard 'linear' form which is construed as follows: Names of sorts (not necessarily written in capitals) are included in parentheses. Sort(s) forming the value type are written first (in case of attribute's being multivalued the name of the value type is written in plural, i.e. with '-s', in case of the value type being a tuple the name of the tuple is written first followed by '=' and the names of particular components of the tuple in parentheses), followed by the sorts forming the arguments. Entity sorts are provided with the sign '#'. Between (and as the case may be before the first and after the last) particular sort names texts (in lower case letters) explaining the semantics of the attribute (the content of „bubbles") are written. The whole name must form a grammatically correct expression of a natural language, which serves at the same time as a control of the 'correctness' of the name.

R4) An attribute can be provided with the so-called *ratio* of the attribute that mirrors the appropriate singular and total integrity constraints. The ratio also determines whether an attribute can serve as an identifying attribute. The ratio is written in the form

***p,m : q,n***

where

- ***p* = 0** means that the attribute is a ***partial function***
- ***p* = 1** means that the attribute is a ***total function***
- ***m* = 1** means that the attribute is ***singular***
- ***m* = M** means that the attribute is ***multivalued***.

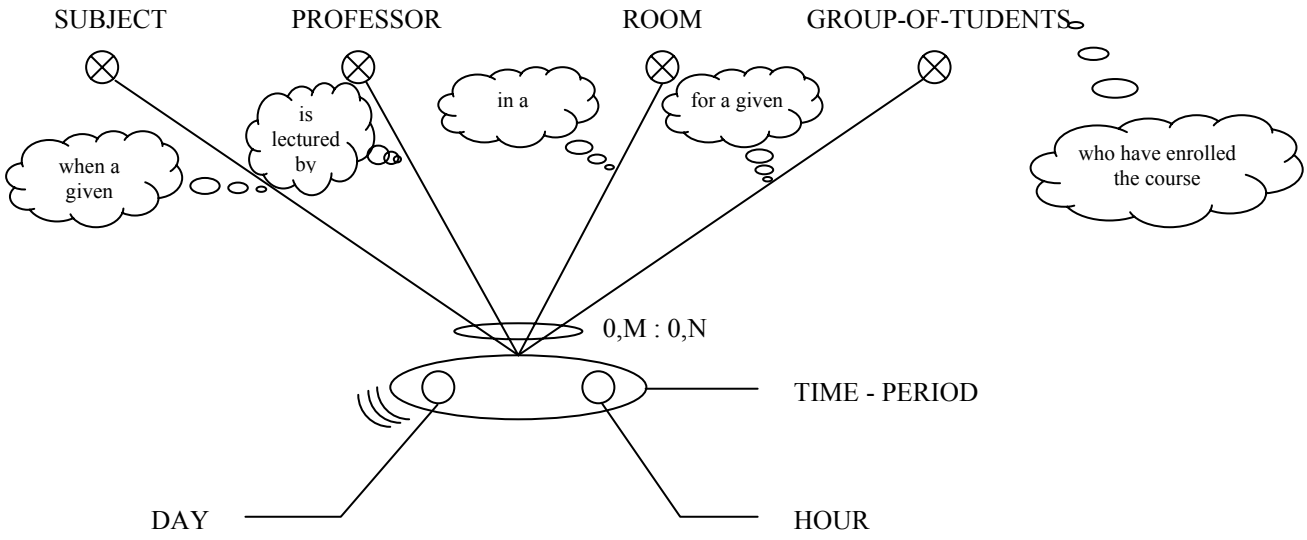
The values of *q* and *n* express these constraints for the *'inverse' function*.

(The term total (partial) function is used here rather non precisely: We mean the fact that the extensional function which is the value of the attribute is in all the states of affairs total (partial). A total function assigns to each element of the argument type just one element of the value type (which is in case of a multivalued attribute a *non-empty* set). A partial function assigns at most one value (in case of a multivalued attribute it may be an empty set).)

An *identification attribute* has the ratio of the form 1,1 : 0,1, i.e. it is a total singular function. There may also be a set of identification attributes (identifying the elements of the argument type) each of which has the ratio 1,1 : 0,M. Note that elements of the argument type are identified by *attributes* not by values.

*Example*: An attribute 'Schedule' expressing the schedule of lectures with a standardised name: (TIME-PERIOD)-s = (DAY, HOUR) when a given (#SUBJECT) is lectured by a given (#PROFESSOR) in a given (#ROOM) for a given (#GROUP-OF-STUDENTS) who have enrolled the course / 0,M:0,N

is graphically depicted by the following A-schema:




The respective construction corresponding to this schema and name is as follows:

$\lambda\, s\, p\, r\, g\, \lambda\, (d,h)\, [[^{0}\text{Lecturing}\ (s, p, r, g)]\ (d, h)]$

(types of variables: $s$ / SUBJECT, $p$ / PROFESSOR, $r$ / ROOM, $g$ / GROUP-OF-STUDENTS, $d$ / DAY, $h$ / HOUR).

*Note* that one of the 'bubble' texts expresses the attribute body.

The goal of the dialogue between a user and a designer is building up a conceptual schema describing the universe of discourse. This work consists in the specification of attributes that are of user's interest and consistency constraints connected with these attributes. Together with recording attributes we also create the base of sorts. Distinguishing entity and descriptive sorts in this early phase is not important yet. Anyway, concerning entity sorts we have to keep in mind that each entity sort has to be precisely *defined*, i.e. the property specifying the sort has to be described, and it has to be *identified* (by an identification attribute or by a set of identification attributes). At the same time relations *subtype–supertype* (see the following Section 3.2.6) between entity sorts have to be defined.

The result of the reality–mapping phase is the first proposal of the HIT-conceptual schema. In the successive phase the designer has to perform all the necessary adjustments of the schema, which consist in definite description of the base of sorts (see above), and mainly in determining the so-called *data kernel*, i.e. transforming our attributes to the simplest, most elementary possible form and specification of informationally redundant attributes. This process will be described in detail in Chapter 4. In practice, of course, both the phases do not have to be successive, they may go (and often do go, especially in case of a skilful designer) in parallel.

### 3.2.6. Traditional modelling constructs and the HIT data model

In this section we sum up particular 'traditional' constructs used in semantic database models and outline the way they are covered by means of the HIT methodology. At the same time we provide a precise logical explication of these constructs.

**I. Sorts** (types)

(Entity and descriptive) sorts have been defined in Section 3.2.2. For the sake of completeness we just briefly recapitulate:

a) **Entity sorts** (abstract types)

This category is usually defined vaguely, the set of entities as being the set of objects of the real world the data of which are recorded in the database. In HIT data model an entity sort E is defined as the set that is given solely by a property (of objects of a type $\alpha$), say P [Materna 1987]. It is the union of classes which are selected by this property through a relevant time interval, i.e. the set of all objects which possessed, possess and will possess the property P. Particular classes selected by P are called *populations* of the entity sort E. The fact that an entity sort is given solely by a property, i.e. an intension (mapping from possible worlds and time points to the type $\alpha$) has an important consequence: It is not a recursive set and it is not representable ('printable'); there is no recursive injection of the sort E into a set of finite strings over a finite alphabet. If an entity sort E were representable, we would have to be able to use a recursive function taking the property P as its argument and returning the class generated by P in the actual world (the population of E). But since we can never know which of the possible worlds is the actual one, no such function can be found.

b) **Descriptive sorts** (printable types)

A descriptive sort is a 'normal' recursive, i.e. representable set. Members of descriptive sorts are encoded by data stored in a database by means of which we describe entity sorts, i.e. they are ranges of functions defined by (descriptive) attributes in particular states-of-affairs. The impossibility of finding a recursive function determining the population of an entity sort, as well as the intensional character of attributes have thus to be compensated by cognitive actions (data collection).

c) **Relationship sorts** (relationship sets)

A relationship set R is actually an n-tuple $(E_1,...,E_n)$, i.e. the Cartesian product $E_1 \times ... \times E_n$ of the sorts $E_1$, …, $E_n$. In practice it often stems from some event (e.g. a loan, returning, lecturing, an order, etc.). It is one of the main building blocks of the Entity-Relationship (E-R) data model [Chen 1976]. Using E-R model is mostly considered to be an adequate tool for business data analysis. But when the business reality is completely unknown to the designer or too complex, the design of relationship sets becomes a stumbling block of the E-R data modelling, for these sets are in a way 'unnatural'. Using the HIT method of conceptual analysis, the designer is not forced to determine these objects of a 'higher', more complex type at the very beginning of the design process. He can map them in a very natural way as n-ary ($n \geq 1$) (relationship) attributes, i.e. as (empirical) functional dependencies between more basic types (entity and descriptive sorts). Particular relationship sets are then obtained by automated process of transformation of the HIT conceptual schema into a 'flatter' Chen's like schema, which consists of representing n-ary functions by means of relationship sets (the so-called 'binarization principle' is applied, see [Duží 1999], Section 5.2), enriching the base of sorts by relationship types and reformulation of consistency constraints associated with these complex attributes. Transforming a 'kernel-like' schema, we obtain a schema in the $4^{th}$ normal form which is informationally equivalent [Duží 1992] with the original HIT schema. The process of transformation, i.e. in fact the process of the design phase of the system 'life-cycle' is

thoroughly described in Section 5 of this study. For example transforming the attribute 'Schedule' from the above example, we obtain a relationship set R = (SUBJECT, PROFESSOR, ROOM, STUDENT-GROUP), four (binary relationship) attributes expressing the relations of R to the original entity sorts, and a (binary descriptive) attribute which could be called „the schedule times of R". (Note that in such complex cases it is difficult to find a natural name for the new relationship set.)

**II. Subtypes, supertypes, inheritance** ('ISA relation')

The relation of 'being a subtype' is usually defined extensionally as the set-theoretical inclusion. For instance, it is being said that the set of employees is a subset of the set of persons. Such a relation can be defined only for descriptive sorts. In case of entity sorts the relation is determined by the fact that some properties are not logically independent. We say that an entity sort $E_1$ which is determined by a property $P_1$ *is a subtype* of an entity sort $E_2$ which is determined by a property $P_2$ iff the property $P_1$ necessarily implies the property $P_2$ in the following sense:

$\forall wt\ \forall x\ ([^0P_{1wt}\ x] \supset [^0P_{2wt}\ x])$,

where *w, t, x* range over possible worlds, time-points and individuals (or generally objects of a type $\alpha$), respectively. The consequence of this dependency is the fact that in *all the states-of-affairs* the population of $E_1$ is a subset of the population of $E_2$. Hence in data modelling the ISA relation is a *necessary relation* in contradistinction to the approach applied in artificial intelligence, where a contingent fact is sometimes also considered to be the ISA relation (e.g. Aristotle is a philosopher).

**Generalisation, specialisation**:

The concepts of properties $P_1$ and $P_2$ do not have to be *primitive concepts* (see Section 3.3 or [Materna 1998]) in a given database conceptual system. For instance, the concept of an employee can be defined as 'an individual that is a person who is employed', the concept of a student as 'an individual that is a person who studies in a ...', the concept of a professor as 'an individual that is a person who lectures', and so on. Hence we can say that the concept of an employee contains the concept of a person, the concept of a student contains the concept of a person, the concept of a professor contains the concept of a person (more precise explication of the content of a concept and the relation of intensional containment see Section 3.3.). Generalisation and specialisation are two types of creating a new entity sort. Having entity sorts $E_1,...,E_n$ defined by properties $P_1,...,P_n$ the concepts of which contain concepts of some properties $Q_1,...,Q_m$, we can define a new entity sort E specified by properties $Q_1,...,Q_m$ as a *generalisation* of $E_1,...,E_n$. For example generalising entity sorts CAR, MOTOR-CYCLE, POPPER, BICYCLE we obtain a new entity sort VEHICLE, generalising further VEHICLE with a PLANE, SHIP, BOAT we obtain TRANSPORT-MEANS. It usually holds in case of generalisation that in every state-of-affairs the populations of sorts $E_1,...,E_n$ do not overlap and the set-theoretical union of these populations covers the population of the new sort E. On the other hand, *specialisation* is an 'opposite' way of defining a new entity sort: Having an entity sort E specified by a property P, we define new entity sorts $E_1,...,E_n$ by specifying some important features, possible roles $P_1,...,P_n$ which an individual having the property P may present or which it may lack to present. For instance, having the entity sort BIRD, we can define PENGUIN as a bird that does *not* fly (does *not* have feathers), or WATER-BIRD, DOMESTIC-BIRD, etc. Or, from the entity sort PERSON we can specialise sorts EMPLOYEE, STUDENT, PROFESSOR, RETIRED-PERSON, etc. These sorts may

overlap, and an object may change such roles without changing its underlying or fundamental identity (e.g. a student might become an employee or cease to be an employee without losing his or her underlying identity as a person).

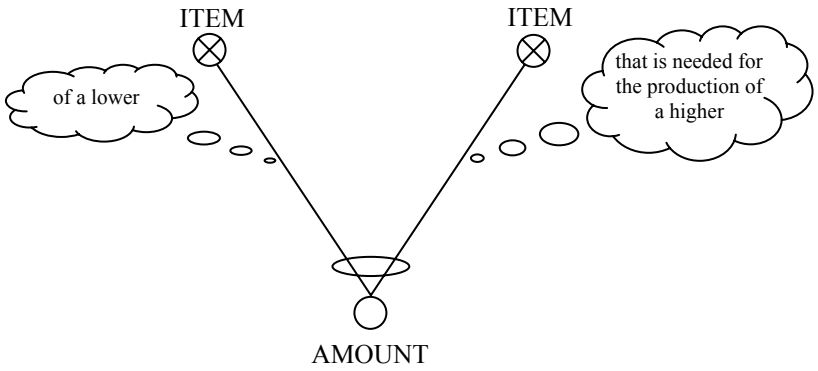
Needless to say that in both cases the sorts $E_1,...,E_n$ are subtypes of the sort E. Moreover, using Kauppi's terminology (that we accept), the concepts of properties $P_1,...,P_n$ are in this case *comparable* (they have something in common).

Both these types of definition are used in the HIT conceptual modelling, but unlike the COMIC system, their using is not promoted in the resulting HIT conceptual schema by special graphical support (besides the ISA relation).

**III. Aggregation** (of sorts $E_1,...,E_n$ into a sort E)

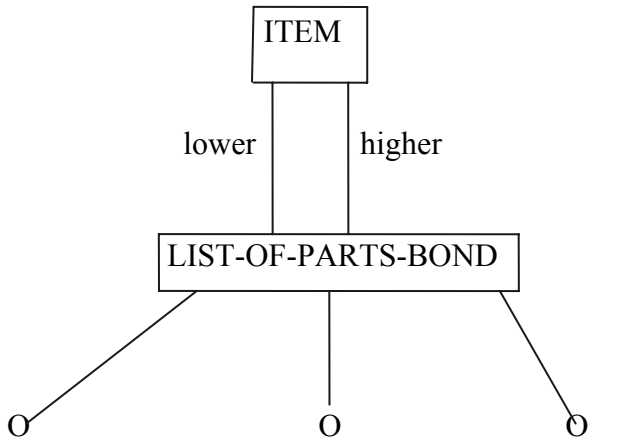
This is actually the part-whole relation. It is either mapped as a tuple (Cartesian product) of (entity) sorts or more frequently by HIT attributes. For example a graph consists of edges and nodes; or a car consists of (is assembled of) a motor, wheels, tires, a bonnet, etc. A typical example is a list of parts (the bill of materials). Each part (production item) consists of (is assembled of) a certain amount of other parts (items) which can again be assembled of other items until the level of basic items (materials) is obtained. Hence each production item is either a final product, or a part (that can be a product as well), or a raw material. This situation is in the HIT method mapped by the attribute with the following name:

'(AMOUNT) of a lower (#ITEM) that is needed for the production of a higher (#ITEM)' with the corresponding A-schema:




If a given item is a final product, it can never occur in the position of a lower item. On the other hand, if a given item is a raw material, it can never occur in the position of a higher item (which is an integrity constraint connected with this attribute).

Transforming this attribute into the E-R schema (see 5.2), we have to introduce a new relationship set, let it call a LIST-OF-PARTS-BOND which is a tuple (ITEM, ITEM), and we obtain the following structure:




IV. **Grouping**

This is in fact a set construct. In the HIT methodology it is mapped by a multivalued attribute. For instance a department is a set of employees. The corresponding HIT attribute is named by

'(#EMPLOYEE)-s who are members of a given (#DEPARTMENT)'.

**V. Attributes, relationships**

In traditional semantic data models only attributes which we call descriptive ones, i.e. attributes of a type E → D (E an entity sort, D a descriptive sort), are used. Relations between entities are expressed by relationships. Our notion of attribute is a broader one than traditionally used. Since HIT attributes are of the so-called *simple types*, see above, (which is valid only for the basic conceptual schema; attributes of derived schemata — views — are even of more complicated types [Zlatuška 1986]), our notion of attribute covers a tuple construct (aggregation), a set construct (grouping), and moreover, it covers relationships between entity sorts as well. This feature makes the model extremely stable. Moreover, our HIT methodology provides us with a logically precise characterisation of the notion of attribute. It is an empirical function—an intension, i.e. a mapping from possible worlds and time points to the set of (analytical) functions of the respective type. We also distinguish between the so-called '*kernel'* attributes and definable (redundant) attributes [Duží 1992], see Section 4 below.

### *3.3. Hit data model from the conceptual point of view*

In this section we first briefly summarise basic notions of the theory of concepts based on the Transparent Intensional Logic (TIL) [Tichý 1988] as they are presented in [Materna 1998]. Afterwards we make more accurate the theory of HIT database conceptual schema from the conceptual point of view; in other words we present a correction of the view presented in [Duží 1999], and of the view presented above, namely of the base of sorts and the type system. We will make a slight simplification of Materna's theory: A *concept* will be taken here to be a closed *construction,* i.e. a concept* of [Materna 1998], where a concept is a *class* of quasi-identical closed constructions, i.e. of such closed constructions that are

indiscernible from the conceptual point of view (for instance in a natural language we cannot find different expressions for quasi-identical constructions). The important features of our conception of concepts are the following:

- Concepts are objective, independent of a particular language;
- Concepts are 'abstract procedures' — sequences of some steps that lead to the identification of an entity (or sometimes fail to identify any — in case of strictly empty concepts); they are timeless and spaceless;
- Concepts are structured non set-theoretical entities: *complexes*; they consist of parts, but the *way of composing* these parts together is important, not only the parts themselves;
- People *do not create concepts*, they just *discover* them;
- Concepts 'exist' independently of our knowledge of them. Concept acquisition is another problem that is not dealt with here;
- 'The meaning of a concept changes' is a non-reasonable sentence, since concepts are meanings of expressions; only the assignment of certain concepts to particular expressions can change.

We can see that the above criteria are actually met by closed constructions. This is the reason for conceiving concepts — meanings of expressions — as closed constructions. What follows is a summary of main definitions concerning Materna's concept theory:

A *simple concept* is a construction ${}^0X$, where X is a variable (of any type) or an object that is not a construction.

The *primitive content (intension) of a concept C* is the set of simple concepts that are subconstructions of C. The primitive content of a simple concept is a singleton.

The *extension* of *a concept C* is the object constructed by C. If the object constructed by C is an intension ($(\omega\tau)\rightarrow\alpha$ - object), i.e. a mapping from the set of possible worlds and time points to a type $\alpha$, we speak also about an *extension of C w.r.t. a world W and a time point T* which is the value of this intension in W at T.

*Note*: The term 'intension' is used here with two different senses: either as an intension of a concept or an intension as a mapping from possible worlds ...( $(\omega\tau)\rightarrow\alpha$ –object, for some type $\alpha$). We will therefore prefer the term 'content' of a concept to the term 'intension' of a concept not to cause any confusion. When analysing natural language expressions in the following examples we will, of course, do that over the epistemic base: $\{o, \iota, \tau, \omega\}$.

*Example*: The concept of an UNMARRIED MAN is not simple:

$$\lambda w \lambda t\, \lambda x\, [{}^0\wedge\, [{}^0\mathrm{Man}_{wt}\, x]\ \ [{}^0\neg\, [{}^0\mathrm{Married}_{wt}\, x]]],$$

where variables *w, t, x* range over the set of possible worlds, time points and individuals, respectively. The content of this concept is the set $\{{}^0\wedge, {}^0\neg, {}^0\mathrm{Man}, {}^0\mathrm{Married}\}$. The extension of this concept is the *property* of being an unmarried man, i.e. an $(\omega\tau\rightarrow(\iota\rightarrow o))$–object. The extension of this concept w.r.t. a world W and a time point T is the set of men that are unmarried (in this W and T).

Now we need to define the notion of conceptual system: Let $C_1,...,C_m$ be simple concepts. Let $C_{m+1},...$ be all the concepts distinct from $C_1,...,C_m$ such that the subconstructions of $C_{m+i}$, $i > 0$, are only members of $\{C_1,...,C_m\}$ and variables ranging over those types that are composed of types given by $C_1,...,C_m$. The set $\{C_1,...,C_m\} \cup \{C_{m+1} ...\}$ is called a *conceptual*

*system (CS)*. The set $\{C_1,...,C_m\}$ is the set of *primitive concepts of CS (PCS)*, the set $\{C_{m+1}$ ...$\}$ is the set of *derived concepts of CS (DCS).*

The last notion that we will need is the notion of *definition*. Let CS be a conceptual system. Let C be a member of DCS, and let C construct an object A. If A is not constructed by a member of PCS, then *C defines A.* An object A is *definable* in the conceptual system CS iff some member of CS defines A.

The following claims are obvious: Every non−strictly empty complex concept (i.e. a concept that is not simple and it does not fail to construct an object) defines some object in some conceptual system. Every not strictly empty complex concept is a *definition* in some conceptual system. Well, this is not a standard use of the term 'definition'. One of the most striking distinctions between the above and the standard one is the fact that there is no 'definiendum' and 'definiens' here. But having explicated the semantic character of definitions, we can define 'linguistic definitions' as expressions of a (sub)language having the form *Definiendum = Definiens* as follows:

Let CS be a conceptual system based on PCS = $\{C_1,...,C_n\}$. A *language $L_{CS}$ of CS* is a language satisfying the following conditions:

a) There are simple expressions in $L_{CS}$ that represent $C_1,...,C_n$.

b) There is a grammatical rule (or a set of such rules) of $L_{CS}$ that makes it possible to create an expression representing composition from expressions representing particular „components" of the composition.

c) If $E_X$ represents a construction X then there is a grammatical rule (or a set of such rules) that makes it possible to create an expression $E_\lambda$ that represents an abstraction $[\lambda x_1...x_m\ X]$.

The language $L_{CS}$ may not contain any 'linguistic definition'. Now we can build up a hierarchy of languages each of which contains some new simple expressions introduced by means of a linguistic definition into the previous level.

i) Let $L_{CS0}$ be $L_{CS}$.

ii) Let $L_{CSi}$ ($i > 0$) result from $L_{CSi-1}$ by adding a set of simple expressions $SE_{i1}$, ..., $SE_{ik}$ ($k > 0$), together with expressions interpreted as true sentences
$SE_{i1} = CE_{i1}$
…
$SE_{ik} = CE_{ik}$
where $CE_{i1}$, ..., $CE_{ik}$ are complex expressions that contain only expressions occurring in lower level languages. The expressions of the above form can be called *linguistic definitions* expressed by the language $L_{CSi}$, $SE_{i1},...,SE_{ik}$ are *definienda*, $CE_{i1},...,CE_{ik}$ are *definiens* expressed by $L_{CSi}$.

Of course, this hierarchy is rather artificial, and natural languages do not develop in such a schematic way, yet it illustrates the connection between conceptual systems and expressions. It is important to realise that a simple expression does not have to represent a simple concept, i.e., a primitive concept of a given CS. It may be an expression in some (sub)language $L_{CSi}$ in which it is defined by means of simpler expressions from $L_{CSi-1},...$ until reaching the level of $L_{CS}$. For instance, a bachelor can be defined as an unmarried man and can thus represent the complex concept

$$\lambda w\ \lambda t\ \lambda x\ [{}^0\wedge\ [{}^0\mathrm{Man}_{wt}\ x]\ \ [{}^0\neg\ [{}^0\mathrm{Married}_{wt}\ x]]]$$

that contains, among others, the concept $^0$Man and the concept $^0$Married; due to this convention (a linguistic definition) a 'bachelor' and an 'unmarried man' are synonymous expressions [Duží 1996] (with respect to the given CS), for they represent one and the same concept. (In this example we suppose that 'man' is connected with a primitive concept of our CS: $^0$Man. In another CS it might happen that the concept of man would be derived from two other primitive concepts of human being and of male. Then 'man' would be connected with a derived concept

$\lambda w \lambda t\, \lambda x$ ([$^0$Human$_{wt}$ $x$] $\wedge$ [$^0$Male$_{wt}$ $x$]) the content of which would be {$^0$Human, $^0$Male, $^0\wedge$}.)

To be able to precisely explicate the relation of intensional containment we have to adjust the definition of the *intension of a concept*, for if such a relation where based on the intension of a concept as defined above, it would be neither reflexive, nor transitive, which is obviously not a desirable state. Therefore we define:

**Definition:**
The *content of a concept C* is the set of subconstructions of C that are themselves concepts.
A concept C *intensionally contains* a concept C' (denoted C ≥ C') iff C' is a member of the concept C.

The relation of intensional containment (IC) defined in this way is reflexive, antisymmetric and transitive, which follows from the definition of a subconstruction:

**Definition:** Let C be a construction.

i) C is a *subconstruction of* C.

ii) Let C be $^0$X. If X is a construction then X is a *subconstruction of* C.

iii) Let C be [X $X_1 \ldots X_n$]. Then X, $X_1$, …, $X_n$ are *subconstructions of* C.

iv) Let C be $\lambda x_1 \ldots x_n$ X. Then X is a *subconstruction of* C.

v) If A is a *subconstruction of* B, and B is a *subconstruction of* C, then A is a *subconstruction of* C.

vi) Anything is a *subconstruction of* C only due to i) – v).

Now considering our simple 'bachelor example' with linguistic definitions

"Bachelor is an unmarried man",

"Man is a male human being",

we have several possibilities how to analyse them, depending on a chosen conceptual system. Imagine that we consider three conceptual systems with primitive concepts as follows:

$PCS_1$: {$^0$Bachelor, …}

$PCS_2$: {$^0$Man, $^0$Mar(ried), …}

$PCS_3$: {$^0$Male, $^0$H(uman being), $^0$Mar(ried), …}

If we choose, for instance, the system generated by $PCS_3$, the we can define the property of being a bachelor as follows:

$\lambda w \lambda t\, \lambda x\ [^0\neg\, [^0\mathrm{Mar}_{wt}\, x]\ \wedge\ [\, [\lambda w \lambda t\, \lambda y\, ([^0\mathrm{Male}_{wt}\, y] \wedge [^0\mathrm{H}_{wt}\, y])]_{wt}\, x\, ]\, ]$

MAN

BACHELOR

It is easy to see that the result is plausible:

BACHELOR ≥ MARRIED, MAN, MALE, HUMAN, NOT, AND

MAN ≥ MALE, HUMAN, AND

After this brief recapitulation of the theory of concepts we are now going to analyse HIT database conceptual schema from the theoretical point of view. In [Duží 1992, 1999] we have defined the notion of *data kernel K* of a set of attributes *A* constructed by a given database conceptual system as a minimum set of elementary (undecomposable) attributes such that *K* is informationally equivalent [Duží 1992] with *A*. This is a very important notion because data kernel does not contain any informationally redundant attributes (we deal with this notion precisely in Section 4.2), i.e. such attributes that would be *definable* from a subset of *K* in the following sense:

Let A, $A_1$, ..., $A_n$ be attributes of a database conceptual schema. We say that A is *definable* from $\{A_1,...,A_n\}$ iff

$$\exists f\, \forall wt\, (^0\mathrm{A}_{wt} = [f\, (^0\mathrm{A}_{1wt},...,^0\mathrm{A}_{nwt})])$$

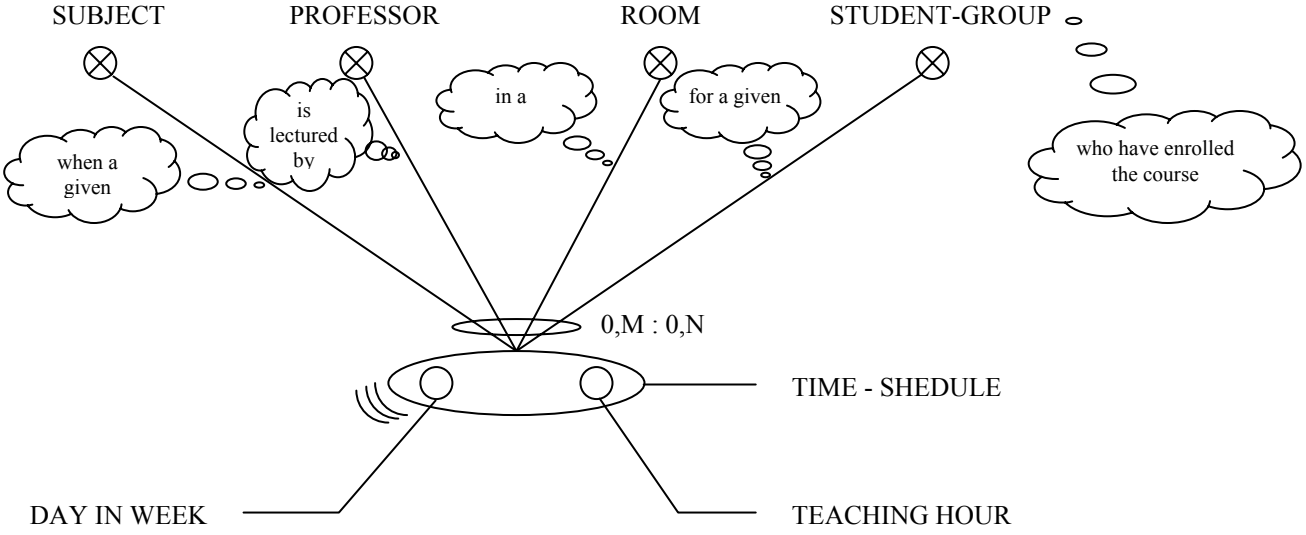
where *f* ranges over (analytical functions) surjections.

Obviously concepts of attributes definable from the data kernel of our database conceptual system are *derived concepts* in the above defined sense (they contain concepts of kernel-like attributes; $^0\mathrm{A} = \lambda w \lambda t\, [^0\mathrm{F}\, (^0\mathrm{A}_{1wt},...,^0\mathrm{A}_{nwt})]$ for some analytical function F) and they should not be a part of the conceptual schema unless there are some special important reasons (effectiveness, reliability). They are normally included in particular external schemata (views). (Moreover, transforming a kernel-like schema into a relational schema we obtain a schema in the 4th normal form, as has been proved in [Duží 1992].)

The problem, however, is whether concepts of kernel-like attributes are primitive concepts of our database conceptual system, as has been affirmed in [Duží 1999]. Consider again the attribute 'Schedule' with the name

> '(TIME-SCHEDULE)-s = (DAY-IN-WEEK, TEACHING-HOUR) when a (#SUBJECT) is lectured by a (#PROFESSOR) in a (#ROOM) for a (#STUDENT-GROUP)'.

Here is the corresponding A-schema:




Writing down the respective construction over the *base of sorts*, we get:

$\lambda\, s\, p\, r\, g\, \lambda\, (d,h)\, [[{}^0\text{Lecturing}\ (s,p,r,g)]\ (d,h)]$,

where variables *s, p, r, g, d, h* are of the types SUBJECT, PROFESSOR, ROOM, STUDENT-GROUP, DAY-IN-WEEK, TEACHING-HOUR, respectively. We come to the conclusion that the content of this concept is a singleton $\{{}^0\text{Lecturing}\}$, hence it is a primitive concept of our conceptual system. (After all, the above construction can be reduced using β-rule to the construction ${}^0$Lecturing.) Isn't it rather unnatural? Wouldn't we say that the concept of this attribute contains concepts of a subject, professor, room, etc. as well? The problem consists in the fact that our type system with the *base of sorts* is not correct. It is just an abbreviation convenient for users. Speaking about the part of reality that is being analysed the user does not employ formulations like „the individuals that have the property of being a professor", „the classes of individuals that have the property of being an organisation" or „the classes of propositions that have the property of being a lecture-course", etc. Instead he/she simply speaks about professors, students, organisations, courses, etc. Indeed, 'student', 'professor', 'organisation, 'course', etc. denote various properties (not necessarily of individuals); the user obviously wishes to express the idea that the information system that is being analysed should offer information about any object that has some of the mentioned properties. This means that the attributes which will be of interest with respect to the information system should concern — and thus be restricted to — only objects with the selected properties. Therefore we use an abbreviation, a convention, and speak about the base of sorts and claim that attributes are functions of simple „types" (i.e. mappings from possible worlds and time points to the set of mappings from n-tuples of sorts to n-tuples of sorts, or to their power set).

There are two flaws in this convention. First, the base of our type system has to be a collection of mutually *disjoint* non-empty sets — elementary types. But sorts are not disjoint ('subtypes' / 'supertypes') and are not elementary. Second, natural language expressions should be analysed (as has been convincingly shown in [Tichý 1988]) over the *epistemic base*: a collection of four elementary types {ο, ι, τ, ω}, where ο is the set of truth values {True, False}, ι is the set of individuals ('naked' entities without any essential properties), τ is the set of time points (or real numbers as their surrogates) and ω is the set of possible

worlds (logically possible states-of-affairs). Indeed, our names of attributes should be analysed over the epistemic base and the above restriction to sorts should be expressed as another condition in the respective construction. Consider, e.g. a very simple attribute 'Salary of an employee'. Using a classic infix notation for logical connectives and the identity sign, it should be precisely analysed over the epistemic base by the following construction:

$\lambda w \lambda t\, \lambda x\, \iota y\, ([{}^0\text{Salary}_{wt}\, x] = y \wedge [{}^0\text{Employee}_{wt}\, x])$,

where $x$ ranges over $\iota$ and $y$ over $\tau$.

Thus the concept of this attribute contains, among others, two primitive concepts (of our CS): ${}^0$Salary and ${}^0$Employee. (The situation is still not as simple, since the simple expression 'employee' is not connected with a primitive concept of our CS but with a derived one.) The above construction constructs an empirical function, namely the attribute that is in all the states-of-affairs defined just for those individuals which are employees and undefined for the others. So far so good.

But in case of a multivalued attribute we would get as a value an empty set, which does not distinguish whether the set is empty because it is actually empty or because the attribute has been applied on a wrong argument(s). Consider, e.g., another very simple attribute 'Children of a given man and a given woman'. Applying the same method as above, we get the construction (variables $x, y, z$ ranging over $\iota$)

$\lambda w \lambda t\, \lambda x\, y\, \lambda z\, ([[{}^0\text{Children}_{wt}\, (x,y)]\, z] \wedge [{}^0\text{Man}_{wt}\, x] \wedge [{}^0\text{Woman}_{wt}\, y])$,

which constructs a function that in all the states-of-affairs returns an empty set either for those couples (man, woman) that do not have any children, or for those pairs of individuals in which the first member is not a man or the second one is not a woman (but we would like to construct such a function which would be undefined for such pairs). Hence we have to make still another adjustment:

$\lambda w \lambda t\, \lambda x\, y\, \iota z\, ([[{}^0\text{Children}_{wt}\, (x,y)] = z] \wedge [{}^0\text{Man}_{wt}\, x] \wedge [{}^0\text{Woman}_{wt}\, y])$,

where $x, y$ range over $\iota$ and $z$ ranges over $(\iota \to o)$, i.e. *sets* of individuals.

Now this construction constructs the empirical function that 'behaves' exactly in the way we wish. The concept of 'Children of a given man and a given woman' contains, among others, the concepts ${}^0$Children, ${}^0$Man, ${}^0$Woman.

This method can be, of course, generalised. The types of empirical functions constructed by particular attribute concepts may be much more complex. Concluding, we can say that the concept of a kernel-like attribute contains concepts of particular sorts and the concept expressing the essence of the attribute, i.e., 'attribute body' (represented usually by one of the texts written in "bubbles" in the A-schema, namely the main one). On the other hand, a concept of a redundant attribute, i.e. attribute definable from the data kernel, contains concepts of particular 'sorts', the concept of the attribute 'body' *and* the concepts of those attributes from which it is definable.

Now we can discuss the problem that can be characterised as the problem of a '*definition of entity sorts*'. It is strictly recommended by the HIT method of database design that each entity sort has to be provided with an exact definition (written in a natural language). For such self-evident sorts like EMPLOYEE, STUDENT, ... it may seem to be dispensable. But there are many other sorts the name of which itself does not sufficiently express its meaning (take, e.g., the sort PRODUCTION-ITEM: is it only an item of a list of parts, or also a final product? may it be also a raw material?); in other words, we use linguistic definitions that assign derived concepts (of our database CS) to simple expressions,

i.e. to names of sorts. Actually, there are many simple expressions which are not connected with primitive concepts of our database CS but with derived ones. Thus, for instance, the sort EMPLOYEE is defined as the set of all individuals that are persons which are employed. The simple expression 'employee' represents a derived concept in the content of which is the concept $^0$Person and the concept $^0$Employed (the respective property is constructed by $\lambda w \lambda t \lambda x$ ([$^0$Person$_{wt}$ $x$] $\wedge$ [$^0$Employed$_{wt}$ $x$])). Since particular traits of the concept of an employee are connected in the *conjunctive* way, it holds that the extension of the concept of employee in every world W and time point T is a subset of the extension of the concept of person in this W and T. We get the "classical" ISA relation. The concept of the attribute 'Salary of an employee' contains concepts $^0$Salary, $^0$Person, $^0$Employed. Expression 'person' is connected with a primitive concept of our CS, whereas the expression 'employee' with a derived one.

Concluding this subsection we summarise: The task of conceptual analysis consists in analysing particular natural language expressions used in the business reality, i.e. in discovering concepts (TIL logical constructions) of entity sorts, descriptive sorts, attributes and integrity constraints that are assigned to these expressions, and in determining relationships between the respective concepts. When performing this task, we should choose as much 'fine-grained' conceptual system, as possible, to reveal all the needed IC relations over our conceptual base. The result of such an analysis is the *HIT conceptual schema.*

### *3.4. Finnish and Czech approaches compared*

In this section two approaches to conceptual data modelling have been compared: the Finnish approach represented by the *COMIC data model* and the Czech approach represented by the *HIT data model*. The basic ideas of the two approaches can be characterised as using *theory of concepts* in both of them, using just the relation of the *intensional containment* (IC) between concepts as the basic modelling construct by the Finnish school, and using the concept of the HIT-*attribute* as the basic modelling construct by the Czech school.

The idea of utilising the IC relation in data modelling is revolutionary and promising. In Kauppi's concept theory this relation is considered to be a primitive pre-theoretical notion. Using our new non-traditional theory of concepts, we have defined this relation in such a way that it fulfils the intuitive criteria stated by Kauppi: it is a reflexive, antisymmetric and transitive relation on the set of concepts.

An attempt to explicate this relation by the COMIC data model is interesting in many aspects. Summarising, we can claim that the classical example of the IC relation is the so-called *ISA relation* between concepts. Having precisely defined the ISA relation by the HIT data model, we can say that only in this case the inverse inclusion relation between extensions of related concepts and the contents (intensions) of the concepts holds in all the possible worlds and time points (due to the conjunctive way of composing particular components 'subconcepts' of concepts).

Another example of the IC relation, the so-called *part-whole relation*, is rather problematic: First, we affirm that only concepts of *necessary* parts of the constructed object can be considered to be contained in the given concept. Second, the inverse inclusion relation between the extensions of the related concepts does not hold in this case. Third, in our opinion, this relation is best modelled again by the concept of a HIT-attribute that can be a partial function, which enables us to catch also the case of *contingent* parts of the constructed object.

In both the above cases (ISA and part-whole) the IC relation is a *partial ordering* on the set of concepts of our CS, i.e. a reflexive, transitive and anti-symmetric relation and the interesting theoretical mathematical results (algebraic properties of the concept lattices, etc.) of [Palomäki 1994], [Niemi 1998, 1999], [Nilsson 1998], [Niinimäki 1999], [Junkkari 1999] can be applied. This cannot be said, however, in case of COMIC-IC considered to model an attribute. In that case problems with contingency of the relation arise, and, moreover, it is *not anti-symmetric*.

The basic modelling construct is perhaps nevertheless the concept of the *HIT-attribute*. Thus the concept of a person does not contain its name, age, address, etc., but the concepts of attributes 'name of a person', address of a person', 'age of a person' contain concepts of person, name, address, age, etc. The concept of an attribute contains concepts of particular sorts, the concept of the attribute 'body' and, as the case may be (redundant attributes) concepts of other attributes as well. This concept enables us to cover not only classical descriptions of entities, but also relationships between entities (*n*-ary attributes, $n \geq 1$), aggregation, grouping and the part-whole relation. Last but not least, using just one modelling construct, HIT-attribute, enables us to use the functional approach with its exact formal apparatus — 'the language of *constructions*' (modified version of the type lambda calculus with tuples), which makes it possible to formally exactly specify not only conceptual schema, i.e. concepts of attributes and consistency constraints, but also particular views and manipulations with data, queries, etc. [Zlatuška 1986].

We have stated above, that from the precise theoretical point of view, attributes should be analysed over the epistemic base. But since the convention (abbreviation) consisting in using the *base of sorts* is very convenient and comprehensive for users as well as for designers of the schema, we will use this convention in the following text and consider attributes as if they were defined over the base of sorts.

# 4. Semantic information connected with data

## *4.1. Informational capability of attributes*

### 4.1.1. Attributes and propositions

The reason for storing values of attributes is that they are able to code semantic information. But data stored in a database are just strings of characters like 'smith', 'eastow', ..., '3000', '8500', ..., and these strings alone provide us with no information at all. Only when knowing the semantics of this data, i.e. when knowing the semantics of the attributes the values of which are encoded in the database, we obtain propositions that, e.g., Mr. Smith's salary is $3000 per month, Mr. Eastow's salary is $8500 per month, etc. And they are just these propositions that are 'bearers' of semantic information. Intuitively, semantic information is connected with the statements of a natural language. It is clear that some statements are less informative than others. Some statements are even not informative at all. For example, the statement

($S_1$) *It is raining in Prague*

is less informative than the statement

($S_2$) *It is raining in Prague and it is foggy weather in Brno*

On the other hand, the statement $S_1$ is more informative than the statement

($S_3$) *It is raining in Prague or it is foggy weather in Brno*

And finally, the statement

($S_4$) *It is raining in Prague or it is not raining in Prague*

provides us with no information at all.

The semantic explanation of these intuitions becomes clear when we realise that statements of a natural language denote propositions, i.e. objects of the type $(\omega\tau \rightarrow o)$.

*Note*: For the sake of simplicity trivialization ($^0$) will be omitted, for in the following considerations we won't need to distinguish concepts of attributes from the attributes themselves. We will now standardly use variables $w$, $w_1$, ... as ranging over states-of-affairs.

We will say that a proposition $p$ *admits* states-of-affairs $w_1,...,w_n$ iff $p$ is true in $w_1,...,w_n$. The above intuitions could now be theoretically explained as follows: a proposition should be the more informative the less probable it is, i.e. the less states-of-affairs it admits [CARN52]. But there are two flaws in this explanation. First, following this theory consequently, we would come to the conclusion that the proposition FALSE (i.e. the function which takes the value False in all the states-of-affairs) which admits no possible worlds, bears the greatest information! And, moreover, there is a question whether a proposition which is not true in the actual world is informative. For instance, is the proposition that the Earth is flat informative? Second, comparison based on cardinality is rather peculiar in case of infinite sets. In contrast to Carnap's notion the set of possible worlds in TIL is not finite (it is even uncountable). To solve the first problem, we assume that the set of possible worlds and time points in which our propositions are true includes the actual possible world and a reasonable time interval (of

the ‘life’ of an information system). In the database world this assumption of true propositions can be justified: we take into account only correct data. The second problem is hereby solved by means of logical implication, which makes it possible to compare informational capability of attribute sets based on the set theoretical inclusion of generated information.

Now we will examine the connection of data (and attributes) with the statements of a natural language. Data code (extensional) functions which are values of (intensional) attributes in particular states-of-affairs. Knowing the concept of an attribute and the value of this attribute in a particular state-of-affairs W, we can generate a set of propositions. For instance, having attribute ‘Salary of an employee’ and its value in W, i.e., the table <tailor, 3000>, <smith, 8500>, ..., we can generate propositions that Mr. Tailor’s salary is $3000, Mr. Smith’s salary is $8500, ..., which are true in W.

From now on we will use variables and objects of the respective types:

| | |
|---|---|
| $p$ / $(\omega\tau \to o)$ | variable ranging over propositions |
| $w$ / $(\omega,\tau)$ | variable ranging over states-of-affairs |
| $x$ / $T$, $y$ / $S$ | $T$, $S$ sorts or tuples of sorts |
| A / a) $(\omega\tau \to (T \to S)$ | singular attribute |
| b) $(\omega\tau \to (T \to (S \to o)))$ | multivalued attribute |

Formalising the above considerations we define:

**Definition 5.**

Let A be an attribute of a type $((\omega\tau) \to (T \to S))$, $((\omega\tau) \to (T \to (S \to o))$ respectively. *The set of basic propositions* $BP(A)^W$ generated by an attribute A in a state-of-affairs W is defined as follows:

$$BP(A)^W = \lambda p\, \exists x \exists y\, ([[AW]x] = y \wedge p = (\lambda w\, [[Aw]x] = y)),$$

$x$ ranging over $T$, $y$ ranging over $S$, $(S \to o)$ respectively. □

*Example*: An attribute A (of a type $T \to S$) the extension of which in a state-of-affairs W is specified by the table

| $T$ | $S$ |
|---|---|
| $t_1$ | $s_1$ |
| $t_2$ | $s_2$ |
| $t_3$ | $s_3$ |
| etc. | |

generates in W the set of propositions:

$$\{\lambda w([[^0Aw]t_1] = s_1), \lambda w([[^0Aw]t_2] = s_2), \lambda w([[^0Aw]t_3] = s_3), ...\}.$$

Trivial as the above definition may seem, it will enable us to exactly prove that the definability relation (Definition 10) induces informational redundancy of attribute sets, which is one of the main contributions of this work. Because of practical reasons, we want to formulate our information about W not only in terms of basic propositions. Yet we also need to define propositions that are the consequences of basic propositions. To this end we are going to adduce some preliminary definitions.

Below we shall denote the type of propositions $(\omega\tau \to o)$ by $\pi$. Let $P$, $Q$ be of type $(\pi \to o)$, i.e., sets of propositions.

**Definition 6.** *Logical implication* $\Rightarrow$ of type $(((\pi \rightarrow o), (\pi \rightarrow o)) \rightarrow o)$ is defined as follows: *P implies Q (P* $\Rightarrow$ Q; infix notation will be used) iff in all the states-of-affairs in which all the members of *P* are true, the members of *Q* are true as well. □

The $\Rightarrow$ relation is reflexive and transitive. It is not anti-symmetric because if $P \Rightarrow Q$ and $Q \Rightarrow P$ then members of *P, Q* are true in the same states-of-affairs but they do not have to be false or undefined (partial functions!) in the same states-of-affairs. We define an abstract class *SP* of *semi-identical propositions* as the set of propositions that are true in exactly the same states-of-affairs and among these states-of-affairs the actual one is included (the assumption of correct data). The $\Rightarrow$ relation now induces a partial ordering on the set of classes *SP,* and we can claim that this is the ordering with decreasing information from left to right. All the sequences converge to the class {TRUE} on the right hand side which does not bear any information at all.

**Definition 7.** *Entailment Cn* of the type $((\pi \rightarrow o) \rightarrow (\pi \rightarrow o))$ is a function that associates any set of propositions *P* with the set of all the logical consequences of *P*:

$$Cn = \lambda p\, [\cup \lambda q\, (p \Rightarrow q)],$$

where *p, q* are variables of the type $(\pi \rightarrow o)$, $\cup$ is the set theoretical union of sets of propositions, i.e., a $(((\pi \rightarrow o) \rightarrow o) \rightarrow (\pi \rightarrow o))$ - object. □

*Note*: Obviously, the operation *Cn* is idempotent, i.e., $[Cn[CnP]] = [Cn\ P]$. Moreover, $P \Rightarrow Q$ iff $[Cn\ Q] \subset [Cn\ P]$; here $\subset$ is the relation of being a subset, i.e., a function of type $(((\pi \rightarrow o), (\pi \rightarrow o)) \rightarrow o)$.

Summarising these considerations we can claim that a set of propositions *P* is more informative than a set *Q* iff *P* implies *Q,* and not vice versa, or iff the set of all the logical consequences of *Q* is a proper subset of all the logical consequences of *P*.

**Definition 8.** *Informational capability of an attribute* A in a state-of-affairs W is the set of propositions $P(A)^W$ generated by A in W, i.e., the set of all the logical consequences of basic propositions generated by A in W: $P(A)^W = [Cn\ BP(A)^W]$.

*Informational capability of a set of attributes* $\{A_1,...,A_n\}$ in a state-of-affairs W is the set of all the logical consequences of the propositions generated by $\{A_1,...,A_n\}$: $[Cn \cup_{i=1}^{n} P(A_i)^W]$. □

*Note*: Obviously, in case of $n = 1$ the informational capability of {A} in W equals $P(A)^W$.

One of the principles to be obeyed when storing data is to avoid redundant data storage unless there are some special reasons (like effectiveness or reliability). But what does the redundant data storage exactly mean? Now we are prepared to define this notion.

**Definition 9**. A set of attributes $A = \{A_1,...,A_m\}$ is *informationally redundant* with respect to a set of attributes $B = \{B_1,...,B_n\}$ ($A \leq_i B$) iff in every state-of-affairs W the informational capability of *A* is less than or equal to the informational capability of *B*, i.e.:

$$\forall w\, [Cn \cup_{i=1}^{m} P(A_i)^w] \subset [Cn \cup_{i=1}^{n} P(B_i)^w].$$

Sets of attributes $A = \{A_1,...,A_m\}$, $B = \{B_1,...,B_n\}$ are *informationally equivalent* ($A =_i B$) iff they have the same informational capability in every state-of-affairs W:

$$\forall w\, [Cn \cup_{i=1}^{m} P(A_i)^w] = [Cn \cup_{i=1}^{n} P(B_i)^w].$$ □

### 4.1.2. Definability of attributes.

To be able to determine whether two sets of attributes are informationally comparable we will now introduce the relation of definability that has been mentioned in Section 3.3, and show that this relation induces informational redundancy of attribute sets. Intuitively, an attribute A is definable over an attribute B if there is an algorithm enabling us to compute in every state-of-affairs *w* the extension of A (i.e. [A*w*]) from the extension of B (i.e. [B*w*]). Indeed, we would obviously say that if (and only if) there is a function which allows us to compute values of an attribute A on the basis of the values of an attribute B, then what can be said (about the world) in terms of A can also be said in terms of B so that the information connected with A is a part of the information connected with B, and A is redundant with respect to B. To simplify our considerations about algorithms (i.e. effective constructions of functions) defined on the extensions of attributes, we will state an assumption that extensions of attributes are finite tables. This is justified by our taking into account a finite discrete time interval of the „life" of an information system.

*Note*: From now on, we will use variable *w* as ranging over states-of affairs ($\omega\tau$), and abbreviate the application $[^0 A w]$ as $A_w$ for any attribute A.

**Definition 10.** An attribute A *is definable over a set of attributes* $\{B_1,...,B_n\}$,

$A \leftarrow_D \{B_1,...,B_n\}$,

iff $\exists f \forall w\ (A_w = [f\ (B_{1w},...,B_{nw})])$,

where *f* ranges over surjections.

A set of attributes *A is definable from* a set of attributes *B* ($A \leftarrow_D B$) iff every member of *A* is definable over a subset of *B*.

Finally, sets of attributes are *mutually definable* ($A \leftrightarrow_D B$) iff $A \leftarrow_D B$ and $B \leftarrow_D A$.

An *attribute* A is said to be *definable from an attribute* B (*mutually definable*) iff $\{A\} \leftarrow_D \{B\}$ ($\{A\} \leftrightarrow_D \{B\}$). □

*Examples*: Let PERSON be the entity sort of persons.

a) Attribute
A = '(NUMBER) of children of a given (#PERSON)'
is definable from attribute
B = '(#PERSON)-s who are children of a given (#PERSON)'
The respective function *F* is defined as follows:
$\lambda b\ \iota a\ (\forall p\ ([ap] = [Cardinality\ [bp]]))$.

Indeed, it holds for all *w*: $[F\ B_w] = \iota a\ (\forall p\ ([ap] = [Cardinality\ [B_w\ p]])) = A_w$.

*Gloss*: Types of extensions: $A_w$ / (PERSON $\rightarrow \tau$), $B_w$ / (PERSON $\rightarrow$ (PERSON $\rightarrow$ o)), variables *a, b, p* of types (PERSON$\rightarrow\tau$), (PERSON$\rightarrow$(PERSON$\rightarrow$o)), PERSON, respectively. *Cardinality* is the function that associates a set with the number of its elements.

b) Attributes
A = '(#PERSON) who is a superior of a given (#PERSON)',
B = '(#PERSON)-s who are subordinates of a given (#PERSON)'

are mutually definable: $A \leftrightarrow_D B$.

Indeed, $\exists f\, \forall w\, (A_w = [f\, B_w])$ and $\exists g\, \forall w\, (B_w = [g\, A_w])$.

The respective functions F, G are defined as follows:
$F = \lambda b\, \lambda p\, \iota q\, ([[bq]p])$, $G = \lambda a\, \lambda p\, \lambda q\, ([aq]=p)$.

Variables:
$a$ / (PERSON→PERSON), $b$ / (PERSON→(PERSON→o)), $p$ / PERSON, $q$ / PERSON

c) Attributes:
A = '(#PERSON)-s who are parents of a (#PERSON)',
B = '(AGE) of a (#PERSON)',
C = 'Average (AGE) of parents of a (#PERSON)';

$C \leftarrow_D \{A, B\}$.

Indeed, $\exists f\, \forall w\, (C_w = [f\, (A_w, B_w)])$, where the respective function F is defined as follows (variables $a$/(PERSON→(PERSON→o)), $b$/(PERSON→τ), $p$/PERSON, $q$/PERSON, $n$/τ, Ave is the function 'Average'):

$F = \lambda ab\, \lambda p\, [\text{Ave}\, \lambda n\, \exists q\, (n=[bq] \wedge [[ap]q])]$.

Now we can prove that a set of attributes $A$ is informationally redundant with respect to a set of attributes $B$ iff $A$ is definable from $B$, i.e. $A \leq_i B$ iff $A \leftarrow_D B$.

**Lemma**. An attribute A is definable over a set of attributes $B_1,...,B_n$ ($A \leftarrow_D \{B_1,...,B_n\}$) iff in all the states-of-affairs $w$ it holds that $P(A)^w \subset [Cn \cup_{i=1}^n P(B_i)^w]$, i.e. iff

$$\cup_{i=1}^n P(B_i)^w \Rightarrow BP(A)^w.$$

*Proof* :

*if*) Let for every $w$ $\cup_{i=1}^n P(B_i)^w \Rightarrow BP(A)^w$. If there were no analytical function F such that $A_w=[F(B_{1w},...,B_{nw})]$ then A and $B_1,...,B_n$ would be logically independent. This means that there would have to be a pair $W_1$, $W_2$ such that all the members of $\cup_{i=1}^n P(B_i)^w$ as well as $BP(A)^w$ would be true in $W_1$, whereas the members of $\cup_{i=1}^n P(B_i)^w$ would be true in $W_2$ and some members of $BP(A)^w$ would be false in $W_2$, which contradicts the assumption.

*only if*) Let there be a function F such that $A_w = [F(B_{1w},...,B_{nw})]$ for every $w$. Let every member of $\cup_{i=1}^n P(B_i)^w$ be true in a state W. If there were a proposition generated by A in this W which were not true, then it would mean that the function F defining A from $B_1,...,B_n$ would have to depend on $w$, which contradicts the assumption.

**Statement 1.** Let *A, B* be sets of attributes. Then $A \leq_i B$ iff $A \leftarrow_D B$.

*Proof*: Follows from the previous lemma and the idempotency and monotony of the operation *Cn*.

**Corollary**. Let *A, B* be sets of attributes. Then $A =_i B$ iff $A \leftrightarrow_D B$.

Statement 1 explicates the intuition introduced at the outlet of this section. Informational comparability of attribute sets is based on the definability relation. Since our notion of attribute is a general one, this result makes it possible to compare informational capability of different data structures, e.g., of a flat and a nested relation. To adduce an example, consider attributes A and B, where A is a nested relation and B is a flat relation:

A = '(#SUPPLIER)-s who deliver a (#MATERIAL)'

B = 'DELIVERERS'.

*Types*:
$A/(\omega\tau \to (MATERIAL \to (SUPPLIER \to o)))$; $B/(\omega\tau \to ((MATERIAL, SUPPLIER) \to o))$.

It is easy to show that B is informationally weaker than A, since $B \leftarrow_D A$ but not vice versa:

$$B = \lambda w\, \lambda ms\, [[A_w\, m]s].$$

However, the attribute C constructed by

$$\lambda w\, \lambda m\, \lambda s\, [B_w\, (m,s)]$$

is not identical to A, because C does not yield materials connected with the empty set of suppliers. Only if there were a consistency constraint connected with the attribute A, namely 'For each material there is always a supplier' ($\lambda w \lambda t\ \forall mat\ \exists sup\ [[A_{wt}\ mat]\ sup]$), then this constraint would ensure the informational equivalence of the attribute A with the flat relation DELIVERERS.

### 4.1.3. Distinguishing capability of attributes

On the set of attributes that „share their domain", i.e. that are such empirical functions the values of which are in all the state-of-affairs of types $(T \to S_1)$, $(T \to S_2)$, ..., where T is a sort or a tuple of sorts, $S_1$, $S_2$, ... are sorts or tuples of sorts, or power sets of (tuples) of sorts, another relation can be defined, namely the relation of *distinguishing capability*, which is sometimes confused with the relation of definability and (erroneously) connected with the informational content of attributes [Vaníček 1988]. Intuitively, the greater the number of discernible classes of an attribute A, i.e. subsets of the universe of discourse, the members of which A does not distinguish, the greater the „power" of A. Observing, e.g., an attribute that associates every animal with its biological class, we can state that this attribute allows to distinguish between, say, mammals and reptiles but it is of no use when we want to distinguish between two animals belonging to the same class; hence it does not enable us to distinguish, e.g., between two distinct races of a dog. Imagine now two attributes A, B that share their domain in the above sense and that distinguish in every state-of-affairs exactly the same subsets of the type T. A question arises: Is the information connected with A the same as the information connected with B? Or a weaker question: Is the amount of information connected with A the same as the amount of information connected with B? Negative answers to both these questions are proved in [Duží 1990, 1992]. Briefly recapitulating the main ideas of the proofs, we define:

**Definition 11.**
Let $A / (\omega\tau \to (T \to S_1))$, $B / (\omega\tau \to (T \to S_2))$, where T is a sort or a tuple of sorts, $S_1$, $S_2$ are sorts, tuples of sorts, or power sets of (tuples) of sorts, variables *w, x, y* ranging over types $(\omega\tau)$, T, T, respectively):

An attribute A has *greater distinguishing capability* than an attribute B, $A \geq_{dc} B$, iff

$$\forall w\, x\, y\ (([A_w\, x] = [A_w\, y]) \supset ([B_w\, x] = [B_w\, y])).$$

An attribute A has *the same distinguishing capability* as an attribute B ($A =_{dc} B$) iff $A \geq_{dc} B$ and $B \geq_{dc} A$.

To be able to compare the $\geq_{dc}$ relation with the informational capability of attributes, we have to be able to compare the former with the $\leftarrow_D$ relation (due to Statement 1). The following assertions make it possible.

*Claim*: $B \geq_{dc} A$ iff $\forall w\ \exists f\ \forall x\ ([A_w\, x] = [f\, [B_w\, x]])$, where $f / (S_2 \to S_1)$ ranges over surjections.

$B =_{dc} A$ iff $\forall w\ \exists f\ \forall x\ ([A_w\ x] = [f\ [B_w\ x]])$, where $f / (S_2 \to S_1)$ ranges over bijections.

*Proof*: Obvious.

Note that the claim does not state that there is a 'universal' function for all the states-of-affairs *w*. Particular functions can be distinct in different states-of-affairs *w*.

For attributes A, B such that the extensions of A and B share the same domain, i.e., $A / (\omega\tau \to (T \to S_1))$, $B / (\omega\tau \to (T \to S_2)$, a stronger notion of definability can be defined:

**Definition 12.** An attribute A is *strongly definable from* an attribute B, $A \Leftarrow_D B$, iff

$\exists f\ \forall w\ \forall x\ ([A_w\ x] = [f\ [B_w\ x]])$,

where $f / (S_2 \to S_1)$ ranges over surjections.

Attributes A, B are *strongly mutually definable*, $A \Leftrightarrow_D B$, iff $A \Leftarrow_D B$ and $B \Leftarrow_D A$.

*Claim*: Strong definability implies definability, and, consequently, strong mutual definability implies mutual definability.

*Proof*: Obvious.

However, the reversal of this claim obviously does not hold. In other words, definability does not imply strong definability.

*Example*: Let PERSON, EMPLOYEE be entity sorts, SALARY, PERIOD descriptive sorts, PERIOD a couple (YEAR, MONTH).

a) Attribute
A = '(NUMBER) of children of a (#PERSON)'
is strongly definable from attribute
B = '(#PERSON)-s who are children of a (#PERSON)':
$A \Leftarrow_D B$.

*Types of extensions*: $A_w$ / PERSON $\to \tau$; $B_w$ / PERSON $\to$ (PERSON $\to$ o).

The respective surjection in question is the function *Cardinality* / ((PERSON $\to$ o) $\to \tau$) that associates every set of persons with the number of its elements. Indeed,

$\forall wp\ ([^0\mathit{Cardinality}\ [B_w p]] = [A_w p])$, *p* ranging over PERSON.

b) Attribute
A = '(NUMBER) that is an average salary of an (#EMPLOYEE) in the year 1999'
is definable from attribute
B = '(NUMBER) that is a salary of an (#EMPLOYEE) in a (PERIOD) = (YEAR, MONTH)'.

However, A is not strongly definable from B; $A_w = [^0F\ B_w]$. The respective (analytical) function F is defined as follows:

$F = \lambda b\ \iota a\ (a = \lambda e[^0\mathrm{Ave}\ \lambda s\ \exists p\ ((p_{(1)} = 1999) \wedge ([b(e,p)] = s)]$,

*Types*: $A_w$ / EMPLOYEE $\to \tau$; $B_w$ / (EMPLOYEE, PERIOD) $\to \tau$; variables *a, b* range over (EMPLOYEE $\to \tau$), ((EMPLOYEE, PERIOD) $\to \tau$), respectively, *e* ranges over EMPLOYEES, *p* over PERIOD and *s* over $\tau$.

c) Attributes
A = '(#PERSON) who is a superior of a (#PERSON)',

B = '(#PERSON)-s who are subordinates of a (#PERSON)'
are informationally equivalent (A $\leftrightarrow_D$ B), but they are not strongly mutually definable. Knowing a person [$A_w$ $p$] — a boss ($p$ ranging over persons), we cannot compute the set of his/her subordinates. We need the whole table [$A_w$] to compute the table [$B_w$] and vice versa.

Now we can compare distinguishing capability of attributes with their informational capability.

*Claim*: If A $\Leftarrow_D$ B then A $\leq_{dc}$ B, but not vice versa.

*Proof*: $\exists f\, \forall w\, \forall x$ ([$A_w$ $x$] = [$f$ [$B_w$ $x$]]) obviously implies $\forall w\, \exists f\, \forall x$ ([$A_w$ $x$] = [$f$ [$B_w$ $x$]]), but not vice versa.

*Example*: Let A, B be as follows: A = 'town name of an address', B = 'ZIP code of an address', i.e., $A_w$ / ADDRESS $\rightarrow$ TOWN, $B_w$ / ADDRESS $\rightarrow$ ZIP. Then A $\leq_{dc}$ B: $\forall w\, x\, y$ (([$B_w$ $x$] = [$B_w$ $y$]) $\supset$ ([$A_w$ $x$] = [$A_w$ $y$]). But A is not (strongly) definable from B, because there is not a universal function that would enable us to compute a town name from the ZIP code. The assignment of ZIP codes to towns is a contingent empirical matter.

*Corollary*: If A $\Leftrightarrow_D$ B then A $=_{dc}$ B (in this case the respective function F is a bijection), but not vice versa.

Note that the idea of the proof is based on the difference between $\exists f\, \forall w$ ... and $\forall w\, \exists f$ .... If the intensional character of attributes were not taken into account, these claims could not be proved and the two relations, viz. $\leq_{dc}$ and $\leq_i$, might easily be confused as it happened in [Vaníček 1988].

Now we can answer the question on the connection between distinguishing capability of attributes and their informational content. We have proved that informational comparability of attributes is based on the $\leftarrow_D$ relation, whereas distinguishing capability is determined by the $\leq_{dc}$ relation. These two relations contain $\Leftarrow_D$ relation as their common intersection. The $\leq_{dc}$ relation is based on the existence of a function mapping a range of the extension of one attribute to the range of the extension of another one. If this function is a universal one (i.e. the same in all the states-of-affairs) then it establishes the strong definability, otherwise the two attributes are not informationally comparable.

### 4.1.4. Distinguishing capability based on cardinality

One could raise an intuitive objection against our disconnecting the $\leq_{dc}$ relation and information. Imagine two witnesses A and B informing the police in the following situations:

a) Witness A knows the name of the wanted person (attribute A), whereas witness B reproduces the identity card number of the person (attribute B).

b) Witness A states the identity card number of the person (attribute A), and witness B states the birth-date identity number of the person. (The birth-date identity number is a number of the form YYMMDD/SSSS, where YY is the year of the birth, MM is the month of the birth in case of a male person or the month + 50 in case of a female person, DD is the day of the birth and SSSS is the serial number assigned to persons being born on the respective day.)

Ad a) It holds that A $\leq_{dc}$ B and it might seem that the witness B were more informative than witness A. We could claim such a fact if A $\leftarrow_D$ B, which is not the case (name is not computable from identity card number). However, are we able to compare the 'amount of

information' connected with these two attributes? Bearing in mind Carnap's idea [Carnap 1952] of comparing informativeness of our assertions, i.e., the more 'probable' the proposition is the less informative it is (tautology does not provide us with any information), we could try to compare cardinalities of possible ranges of the attributes. But this comparison is not based on the $\leq_{dc}$ relation as will be shown at the end of this section.

Ad b) It might seem that two attributes of the same distinguishing capability provide us with the same amount of information. But this example illustrates the falseness of this assumption. Indeed, the age, the date of birth and the sex of a person can be deduced from the birth-date identity number but not from the identity card number.

Cardinalities of two sets can be, of course, compared even if none of them is a subset of the other. To be able to grasp this situation as well, we define:

**Definition 13**. $A \leq_{bc} B$ (*comparison based on cardinality*) iff

$$[^0\mathit{Cardinality}\ \lambda y\ \exists wx\ ([A_w\ x] = y)] \leq [^0\mathit{Cardinality}\ \lambda z\ \exists wx\ ([B_w\ x] = z)].$$

Similarly $A =_{bc} B$ iff $A \leq_{bc} B$ and $B \leq_{bc} A$.

*Example*: Compare attributes $N_1$ = 'First name of a person' and $N_2$ = 'Surname of a person'. We are again tempted to claim that the former gives less information than the letter. In this case this assertion is justified: The greater informativeness of $N_2$ can be explained by the fact that cardinality of the set of all the usable surnames is greater than the cardinality of the set of all the usable first names, hence $N_1 \leq_{bc} N_2$; to be named Charles, e.g., is certainly more probable than to be named Eastow. (To ensure the correctness we can restrict our considerations to European first names and surnames.) This does not, of course, mean that the less than relation holds between the cardinalities of the ranges of the extensions of these attributes in *all* the states of affairs. In an apocalyptic state of affairs, where only the Smith family has survived and its members have different first names the proposition

$$\lambda w\ ([^0\mathit{Cardinality}\ \lambda y\ \exists x\ ([N_{1w}\ x] = y)] \geq [^0\mathit{Cardinality}\ \lambda z\ \exists x\ ([N_{2w}\ x] = z)])$$

takes the value true.

Comparing (the amount of) informational content of attributes is sometimes confused with comparing '*the value*' of information provided. In case a) of the above example it is highly probable that the information obtained from the identity card number will be more valuable to the policeman than the information obtained from the name. But being on an island where our policeman knows the names of all the inhabitants, and where he does not have a file of identity card numbers, we would agree that the name is more valuable information for him, which, of course, does not contradict the fact that logically it provides less information.

It might seem that if $A \leq_{dc} B$ then $A \leq_{bc} B$. But this does not hold. These two relations are not comparable. The only fact that can be stated is the following: If $A \leq_{dc} B$ then

$$\forall w\ ([^0\mathit{Cardinality}\ \lambda y\ \exists x\ ([B_w\ x] = y)] \geq [^0\mathit{Cardinality}\ \lambda z\ \exists x\ ([A_w\ x] = z)],$$

from which it does not follow that $A \leq_{bc} B$. (This is a correction of [Duží 1990].)

*Example*: Imagine attributes A, B such that A assigns in world $W_1$ values $a_1$, $a_2$, in $W_2$ values $a_3$, $a_4$, in $W_3$ values $a_5$, $a_6$, and so on, whereas B assigns in all worlds *w* only values $b_1$, $b_2$, $b_3$. Then $A \leq_{dc} B$ but $A \geq_{bc} B$.

In summary, speaking about (informationally) redundant attributes is justified only in case of the definability relation. The $\leq_{dc}$ relation is not generally connected with informational capability of attributes. Only a subset of this relation — the strong definability — specifies

(informational) redundancy. The $\leq_{bc}$ relation can be connected only with the amount of information generated from attributes, and it is not comparable with the $\leq_{dc}$ relation.

### *4.2. Data kernel*

There are two interesting problems connected with building up a HIT conceptual (and generally database) schema. The first one can be characterised as the problem of finding the so-called *data kernel* of the set A of attributes of a given schema, i.e. a minimum set K of 'elementary' attributes such that K is informationally equivalent with A. Such a kernel can serve as an invariant of the whole database system. The second important problem is a problem of polarity described in [Hull 1987], i.e. the problem of 'dual viewing' attributes: either as (*n*-ary) empirical functions or as complex (encapsulated) objects. We will show that these two philosophical approaches can be interrelated by means of a key notion of the transformation of a schema, which will be described in Section 5.

Having a database schema $CS = (A, C_A)$, we will aim to minimise the set $A$ of attributes and to simplify particular attributes. The former can be realised by finding a minimum subset of $A$ informationally equivalent with $A$, i.e. by excluding redundant (sets of) attributes from $A$, which is justified by the following Statement 2. The latter is realised by 'decomposing' attributes (Statements 4 and 5).

**Statement 2.** Let $A = \{A_1,...,A_m\}$, $A' = \{A_{m+1},...,A_n\}$ be sets of attributes such that $A' \leftarrow_D A$. Then $A \cup A' =_i A$.

*Proof*: If $A' \leftarrow_D A$ then (Statement 1) $\forall w\ ([Cn \cup_{j=m+1}^{n} P(A_j)^w] \subset [Cn \cup_{i=1}^{m} P(A_i)^w])$, which means that (Definitions 6, 7)
$\forall w\ ([\cup\ \lambda p\ (\cup_{j=m+1}^{n} P(A_j)^w\ \Rightarrow p)] \subset [\cup\ \lambda p\ (\cup_{i=1}^{m} P(A_i)^w\ \Rightarrow p)])$, hence
$\forall w\ ([\cup\ \lambda p\ (\cup_{i=1}^{n} P(A_i)^w\ \Rightarrow p)] = [\cup\ \lambda p\ (\cup_{j=1}^{m} P(A_j)^w\ \Rightarrow p)])$, i.e.,
$\forall w\ ([Cn \cup_{i=1}^{n} P(A_i)^w] = [Cn \cup_{j=1}^{m} P(A_j)^w])$.

Thus by excluding subsets of the set of attributes $A$ which are definable from $A$ we obtain an informationally equivalent set of attributes, we do not lose any information. This process can be repeated as long as a set of attributes $K'$ is obtained such that no subset of $K'$ is informationally redundant.

To obtain 'elementary' attributes we decompose complex attributes into simpler 'subattributes' in such a way that informational capability of the database conceptual schema is preserved. This process will now be formally described.

**Definition 14.** Let A be an attribute of a type
a) $(\omega\tau \rightarrow ((S_1, S_2) \rightarrow T))$ or
b) $(\omega\tau \rightarrow ((S_1, S_2) \rightarrow (T \rightarrow o)))$,
where $S_1$, $S_2$, $T$ are sorts or tuples of sorts, variables $x/S_1$, $y/S_2$, $z/T$ (case a), or $z/(T \rightarrow o)$ (case b).
We will call a *plural subattribute* of A such an attribute $A_1$ which is constructed from A as follows:
$A_1 = \lambda w\ \lambda x\ \lambda y\ \exists z\ ([A_w\ (x,y)] = z)$
A *singular subattribute* of the attribute A is an attribute $A_2$ constructed as follows:
$A_2 = \lambda w\ \lambda x\ \iota z\ \exists y\ ([A_w\ (x,y)] = z)$ □

*Note*: Obviously, a (singular, plural) subattribute of an attribute A is definable from A.

**Definition 15.** An attribute A *is decomposable* if there are subattributes $A_1$, $A_2$ of A such that $\{A\} =_i \{A_1, A_2\}$. □

**Lemma**: Let A = $\lambda w \lambda x \lambda y\ C$ ($C$/o), A' = $\lambda w \lambda x \lambda y\ C'$ ($C'$/o) be attributes which are either identical or A and A' differ only in the following way:
$\forall wx$ ( $[A_w\, x] = [A'_w\, x] \vee ([A_w\, x] = \{\} \wedge [A'_w\, x] = \{\perp\})$ ),
where $\{\}$ is an empty class ($C$ constructs False) whereas $\{\perp\}$ is such a class U that no member of the given type can be said to belong to U owing to the fact that $C'$ is improper.
Then $A =_i A'$.

*Proof*: Follows obviously from Definition 5. The sets of basic propositions generated from A and A' are identical.

*Note*: Attributes A, A' which differ only in the way described in the above lemma will be called *semi-identical*.

**Definition 16.** Let A be an attribute of a type
a) $\omega\tau \rightarrow ((S_1,...,S_k) \rightarrow (S_{k+1},...,S_n))$ or
b) $\omega\tau \rightarrow ((S_1,...,S_k) \rightarrow ((S_{k+1},...,S_n) \rightarrow o))$,
where $S_1,...,S_n$ are sorts. Let $x_1,...,x_n$ be variables of the types $S_1,...,S_n$, respectively, $(i_1,...,i_n)$ a permutation of (1,...,n).

An attribute A' is an *lambda rotation* of the attribute A if A' is constructed as follows:
case a): $\lambda w\ \lambda x_{i1}...x_{ij}\ \lambda x_{ij+1}...x_{in}\ ([A_w\ (x_1,...,x_k)] = (x_{k+1},...,x_n))$
case b): $\lambda w\ \lambda x_{i1}...x_{ij}\ \lambda x_{ij+1}...x_{in}\ ([[A_w\ (x_1,...,x_k)]\ (x_{k+1},...,x_n)])$.

An attribute A'' is a *singular rotation* of the attribute A if A'' is constructed as follows:
case a): $\lambda w\ \lambda x_{i1}...x_{ij}\ \iota(x_{ij+1}...x_{in})\ ([A_w\ (x_1,...,x_k)] = (x_{k+1},...,x_n))$
case b): $\lambda w\ \lambda x_{i1}...x_{ij}\ \iota(x_{ij+1}...x_{in})\ ([[A_w\ (x_1,...,x_k)]\ (x_{k+1},...,x_n)])$.

A singular rotation A'' of the attribute A is called *admissible singular rotation* (ASR) if in all the states-of-affairs $w$ the extension $A''_w$ is undefined just on those arguments $x_{i1},...,x_{ij}$ for which the respective extension $A'_w$ of the lambda rotation A' constructs an empty class. □

Now there is a question whether and when a rotation of an attribute is informationally equivalent with the original attribute. Obviously a singular rotation does not have to be equivalent, but it might seem that a lambda rotation is always equivalent. We are going to adduce a counter-example showing that it is not so. Consider an attribute

S = '(#STUDENT)-s taking lectures of a given (#COURSE) for which a given (#PREREQUISITE) is prescribed',

and its rotation

S' = '(#PREREQUISITE)-s of a given (#COURSE) lectures of which are taken by a given (#STUDENT)'.

In case that the attribute S is a partial function in the sense that the value of S at some arguments is an empty set of students, we can record by means of S also such courses and their prerequisites which are not endorsed by any student. On the other hand, if the attribute S' returns an empty set of prerequisites at some arguments, then we can record by S' all the endorsed courses, i.e., also such courses which do not have any prerequisite. From this fact we can deduce that equivalent rotations can be constructed only for such attributes that are total, i.e., the first number $p$ of attribute's ratio is equal to 1. Thus we formulate the following statement:

**Statement 3**.

i) Let A' be a lambda rotation of a total attribute A. Then $A' =_i A$.

ii) Let A'' be a singular rotation of an attribute A. Then A'' $=_i$ A iff A'' is admissible.

*Proof*: ad i): Obviously A' $\leftrightarrow_D$ A.

ad ii): Denoting

$C = ([A_w\ (x_1,...,x_k)] = (x_{k+1},...,x_n))$ in case a) or

$C = ([[A_w\ (x_1,...,x_k)]\ (x_{k+1},...,x_n)])$ in case b),

$x = (x_{i1},...,x_{ij})$, $y = (x_{ij+1},...,x_{in})$, we get:

$A' = \lambda w \lambda x \lambda y\ C$, $A'' = \lambda w \lambda x\ \iota y\ C$.

It is now sufficient to prove that A' $=_i$ A''.

Obviously A'' $\leftarrow_D$ A': $A'' = \lambda w \lambda x\ \iota y\ ([[A'_w\ x]y])$.

To prove that A' $\leftarrow_D$ A'' we show that the attribute $B = \lambda w\ \lambda x \lambda y\ ([A''_w\ x] = y)$ is semi-identical with A' just in case that A'' is an admissible singular rotation (ASR) of A. Let A'' be not an ASR of A. Then there is a state-of-affairs W such that at least one value $\underline{x}$ of the variable $x$ exists for which the following holds:

$[A''_W\ \underline{x}] = \iota y\ C = \perp$ (undefined),

$[A'_W\ \underline{x}] = \lambda y\ C$ is a non-empty (at least two elements') class,

while $[B_W\ \underline{x}] = \lambda y\ ([A''_W\ \underline{x}] = y)$ constructs the class $U = \{\perp\}$ and B is not semi-identical with A'.

If A'' is an ASR of A then $[A'_w\ x] = \lambda y\ C$ is either a singleton or an empty class for all the states-of-affairs $w$. Hence B is semi-identical with A'.

Now we shall define a condition to be fulfilled in order that an attribute A is decomposable. Intuitively speaking, we will show that an attribute A is decomposable if in all the states-of-affairs $w$ the values of the extensional function defined by A in $w$ do not depend on some of the arguments of this function $A_w$. In such a case we will say that the attribute A satisfies the *condition of proper singularity*. We will first define such a condition for a singular attribute (Statement 4) and then generalise the condition for a multivalued attribute (Statement 5).

**Statement 4.** Let A be an attribute of a type $(\omega\tau \rightarrow ((S_1, S_2) \rightarrow T)$, where $S_1$, $S_2$, $T$ are sorts or tuples of sorts. Let $x, y, z$ be variables of the types $S_1$, $S_2$, $T$, respectively. We construct a lambda rotation A' of A:

$A' = \lambda w\ \lambda x\ \lambda yz\ ([A_w\ (x,y) = z)$.

Let a singular subattribute

$A_1 = \lambda w\ \lambda x\ \iota z\ \exists y\ ([A_w\ (x,y)] = z)$

exist such that in all the states-of-affairs $w$ the extension $A_{1w}$ is defined just for those arguments $x$ for which $[A'_w\ x] = \lambda yz\ ([A_w\ (x,y)] = z)$ constructs a non-empty class. (In such a case the attribute A satisfies the *condition of proper singularity*.) Then the attribute A is decomposable into subattributes

$A_1 = \lambda w\ \lambda x\ \iota z\ \exists y\ ([A_w\ (x,y)] = z)$ and
$A_2 = \lambda w\ \lambda x\ \lambda y\ \exists z\ ([A_w\ (x,y)] = z)$.

*Proof*: We have to prove that $\{A\} =_i \{A_1, A_2\}$. Obviously, $\{A_1, A_2\} \leftarrow_D \{A\}$. To prove that $\{A\} \leftarrow_D \{A_1, A_2\}$ we shall define an attribute B constructed from attributes $A_1$, $A_2$ as follows: $B = \lambda w\, \lambda x\, \lambda yz\, ([A_{1w}\, x]=z \wedge [[A_{2w}\, x]y])$; now we prove that B is semi-identical with A' (which is sufficient because in this case $B =_i A' =_i A$). Consider an arbitrary state-of-affairs W, $\underline{x}$ a value of the variable *x*.

1. If $[A'_W\, \underline{x}] = \lambda yz\, ([A_W\, (\underline{x},y)]=z)$ constructs a non-empty class C of couples $(y,z)$, then the variable *z* takes the same value, say $\underline{z}$, in all these couples (otherwise $[A_{1W}\, \underline{x}]$ would be improper, which contradicts the assumption). Let the class C be $\{(\underline{y}_1,\underline{z}),...,(\underline{y}_n,\underline{z})\}$. Hence
   $[A_W\, (\underline{x},\underline{y}_1)]=\underline{z}$, ..., $[A_W\, (\underline{x},\underline{y}_n)]=\underline{z}$,
   $[A_{1W}\, \underline{x}] = \iota z\, \exists y\, ([A_W\, (\underline{x},y)]=z) = \underline{z}$,
   $[A_{2W}\, \underline{x}] = \lambda y\, \exists z\, ([A_W\, (\underline{x},y)]=z) = \{\underline{y}_1,...,\underline{y}_n\}$,
   $[B_W\, \underline{x}] = \lambda yz\, ([A_{1W}\, \underline{x}] = z \wedge [[A_{2W}\, \underline{x}]y]) = C$.
   Hence $[A'_W\, \underline{x}] = [B_W\, \underline{x}]$.
2. If $[A'_W\, \underline{x}] = \lambda yz\, ([A_W\, (\underline{x},y)]=z)$ does not construct a non-empty class, then $[A_{1W}\, \underline{x}]$ is improper and $[A_{2W}\, \underline{x}]$ constructs an empty class. Thus $[B_W\, \underline{x}]$ constructs $\{\perp\}$, which means that B is semi-identical with A'.

**Corollary 1**: If an attribute A is decomposable into subattributes $A_1$, $A_2$, then any lambda rotation or an admissible singular rotation of the attribute A is also decomposable into $A_1$, $A_2$.

**Corollary 2**: A singular attribute A of a type $(\omega\tau \rightarrow ((S_1, S_2) \rightarrow T))$, where *T* is a tuple of sorts $(T_1,...,T_n)$, is decomposable into subattributes of types

$(\omega\tau \rightarrow ((S_1, S_2) \rightarrow T_1))$,
$(\omega\tau \rightarrow ((S_1, S_2) \rightarrow T_2))$,
…
$(\omega\tau \rightarrow ((S_1, S_2) \rightarrow T_n))$, respectively.

*Note*: If a singular attribute A satisfies the condition of proper singularity, then the following holds: $\forall w\, x\, y_1\, y_2\, z_1\, z_2\, (([A_w\, (x,y_1)]=z_1 \wedge [A_w\, (x,y_2)]=z_2) \supset z_1 = z_2)$.

Now we can generalise the condition of proper singularity for a multivalued attribute.

**Statement 5**. Let A be a multivalued attribute of a type $(\omega\tau \rightarrow ((S_1, S_2) \rightarrow (T \rightarrow o)))$, where $S_1$, $S_2$, $T$ are sorts or tuples of sorts. Let *x* be a variable of the type $S_1$, *y*, $y_1$, $y_2$ variables of the type $S_2$, *z*, $z_1$, $z_2$ variables of the type $(T \rightarrow o)$. Let the following condition hold:

$\forall w\, x\, y_1\, y_2\, z_1\, z_2\, (([A_w\, (x,y_1)]=z_1 \wedge [A_w\, (x,y_2)]=z_2) \supset z_1 = z_2)$.

Then the multivalued attribute A satisfies the *condition of proper singularity*, and it is decomposable into subattributes

$A_1 = \lambda w\, \lambda x\, \iota z\, \exists y\, ([A_w\, (x,y)] = z)$ and
$A_2 = \lambda w\, \lambda x\, \lambda y\, \exists z\, ([A_w\, (x,y)] = z)$.

*Proof*: We have to prove that $\{A\} =_i \{A_1, A_2\}$. Obviously, $\{A_1, A_2\} \leftarrow_D \{A\}$. To prove that $\{A\} \leftarrow_D \{A_1, A_2\}$ we shall define an attribute B constructed from the attributes $A_1$, $A_2$ as follows

$$B = \lambda w\, \lambda x\, \lambda yz\, ([A_{1w}\, x]=z \wedge [[A_{2w}\, x]y]),$$

and prove that B is identical with an attribute $A' = \lambda w\, \lambda x\, \lambda yz\, ([A_w\, (x,y)]=z)$ which is a rotation of the attribute A).

Consider an arbitrary state-of-affairs W, $\underline{x}$ a value of the variable *x*.

Now $[A'_W \underline{x}] = \lambda yz\ ([A_W\ (\underline{x},y)] = z)$ constructs a non-empty class C of couples $(y,z)$ and the variable $z$ takes the same value, say $\underline{z}$, in all these couples. Let this class be $\{(\underline{y}_1,\underline{z}),...,(\underline{y}_n,\underline{z})\}$.

Then $[A_{1W}\ \underline{x}] = \iota z\ \exists y\ ([A_W\ (\underline{x},y)] = z) = \underline{z}$,

$[A_{2W}\ \underline{x}] = \lambda y\ \exists z\ ([A_W\ (\underline{x},y)] = z) = \{\underline{y}_1,...,\underline{y}_n\}$,

$[B_W\ \underline{x}] = \lambda yz\ (\ [A_{1W}\ \underline{x}] = z \wedge [[A_{2W}\ \underline{x}]y]\ ) = C$.

Hence $[A'_W\ \underline{x}] = [B_W\ \underline{x}]$. This fact is valid also in case of $\underline{z}$ being an empty class.

The condition of proper singularity guarantees, in fact, that the value $\underline{z}$ of $A_w$ assigned to a couple $(x,y)$ does not depend on y. In the relational data model (RDM) [Ullman 1988] an analogous condition is called *functional dependency* (denoted $x \longrightarrow z$) in the case of a singular attribute or a *multivalued dependency* (denoted $x \longrightarrow\!\!> z$) in the case of a multivalued attribute. Statements 4 and 5 thus explicate the algorithms of the so-called lossless-join decomposition into Boyce-Codd normal form or 4th normal form, respectively. (Statements 4 and 5 are, of course, formulated for general attributes.) The lossless-join decomposition can be understood in such a way that the informational capability of an attribute A (corresponding to a relation scheme R) is the same as the informational capability of $\{A_1,A_2\}$, where $A_1$, $A_2$ correspond to the relations $R_1$, $R_2$). The exact explication of the correspondence of HIT-attributes to relation schemes can be found, e.g., in [Zlatuška 1986], [Zlatuška 1990].

*Example:* Consider the attribute S = '(SALARY) of a given (#EMPLOYEE) working on given (TASK)' of the type: $(\omega\tau \rightarrow ((\text{TASK, EMPLOYEE}) \rightarrow \text{SALARY}))$.

Assuming that the salary of an employee does not depend on the tasks he is working on, we can deduce that the attribute S satisfies the condition of proper singularity

(EMPLOYEE —> SALARY) with a singular subattribute

$S_1$ = '(SALARY) of an (#EMPLOYEE)',

And thus S is decomposable into two subattributes (variables *s*/SALARY, *t*/TASK, *e*/EMPLOYEE):

$S_1 = \lambda w\ \lambda e\ \iota s\ \exists t\ ([S_w\ (t,e)] = s)$
$S_2 = \lambda w\ \lambda e\ \lambda t\ \exists s\ ([S_w\ (t,e)] = s)$

*Example*: The case of a decomposable multivalued attribute can be demonstrated by the attribute

A = '(#AUTHOR)-s of a given (#BOOK) published by a given (#EDITOR)'

of the type $(\omega\tau \rightarrow ((\text{BOOK, EDITOR}) \rightarrow (\text{AUTHOR} \rightarrow o)))$. Attribute A satisfies the condition of proper singularity (BOOK —>> AUTHOR). Author(s) of a given book do not depend on the editor(s) who have published the book. Attribute A is thus decomposable into subattributes $A_1$ and $A_2$ (variables *a*/AUTHOR, *b*/BOOK, *e*/EDITOR):

$A_1 = \lambda w\ \lambda b\ \lambda a\ \exists e\ ([[A_w\ (b,e)]a])$

$A_2 = \lambda w\ \lambda b\ \lambda e\ \exists a\ ([[A_w\ (b,e)]a])$.

It is important to comprehend the meaning of the attributes obtained by decomposition. For instance the attribute $S_1$ of the above example is not actually the attribute '(SALARY) of an (#EMPLOYEE)' but the attribute '(SALARY) of an (#EMPLOYEE) working on some task(s)'. Hence using the attribute $S_1$ we would not record salaries of those employees who do not work on any task just now. Similarly the attribute $A_1$ of the above

example yields only information about the authors of those books that have been published. Of course, in practice, when designing a database conceptual schema we might wish to replace the attributes $S_1$ and $A_1$ by the attributes $S_1$' = '(SALARY) of an (#EMPLOYEE)' and $A_1$' = '(#AUTHOR)-s of a (#BOOK)' which in actual fact have a broader semantics than those obtained by the decomposition (and for which the equivalence $\{A\} =_i \{A_1', A_2'\}$ does not hold).

These facts are even more obvious when considering the attributes of the following example:

B = '(#STUDENT)-s of a (#COURSE) who have taken a given (#REREQUISITE) for the course' of the type $(\omega\tau \rightarrow ((\text{COURSE, PREREQUISITE}) \rightarrow (\text{STUDENT} \rightarrow o)))$.

This attribute is decomposable into subattributes (variables *c*/COURSE, *p*/PREREQUISITE, *s*/STUDENT)

$B_1 = \lambda w\, \lambda c\, \lambda s\, \exists p\, ([[B_w\, (c,p)]s])$
$B_2 = \lambda w\, \lambda c\, \lambda p\, \exists s\, ([[B_w\, (c,p)]s])$

with the semantics

$B_1$ = '(#STUDENT)-s of a given (#COURSE) who took its prerequisites' and
$B_2$ = '(#PREREQUISITE)-s prescribed for a given (#COURSE) which were taken by the students of the given course',

provided all the students of the given course took all the prescribed prerequisites. The multivalued dependency COURSE —>> PREREQUISITE is in actual fact an attribute with different semantics, say $B_2$' = '(#PREREQUISITE)-s which are prescribed for a given (#COURSE)'.

The problem of the so-called embedded multivalued dependencies [ULLM88] can be solved by a gradual application of Statements 4 or 5. For instance, the attribute

C = 'The (YEAR) in which a given (#STUDENT) took a given (#PREREQUISITE) for a given (#COURSE)'

of the type $(\omega\tau \rightarrow ((\text{COURSE, STUDENT, PREREQUISITE}) \rightarrow \text{YEAR}))$, can be decomposed (using Statement 4) into subattributes (the type of variables obvious)

$C_1 = \lambda w\, \lambda s\, p\, \iota y\, \exists c\, ([C_w\, (c,p,s)]=y)$

$C_2 = \lambda w\, \lambda c\, p\, \lambda s\, \exists y\, ([C_w\, (c,p,s)]=y)$.

The latter can be again decomposed (using Statement 5) in the same way as in the former example. Thus the embedded dependencies of the attribute C (COURSE —>> STUDENT, COURSE —>> PREREQUISITE) ensure the conditions of proper singularity for the attribute $C_2$. Again, we have to bear in mind that the semantics of, e.g., attribute $C_1$ is in actual fact not the attribute 'The (YEAR) in which a given (#STUDENT) took a given (#PREREQUISITE)' but it bears only information about those students who are enrolled in some course (for which the prerequisites have been prescribed).

Note that the condition of proper singularity of an attribute A is, in fact, an *empirical consistency constraint* connected with this attribute. It is a well-known fact in the theory of the Relational Data Model that the lossless-join decomposition into Boyce-Codd (4th) normal form does not necessarily preserve functional dependencies. In such a case we have to transform particular consistency constraints as well. This is achieved by substituting

subattributes $A_1$, $A_2$ of the attribute A for A in the respective consistency constraint connected with A ($CC_A$).

*Example*: Consider the attribute

A = '(#CITY) which is determined by a given (ZIP) code and (#STREET)'

of the type ($\omega\tau \rightarrow$ (ZIP, STREET) $\rightarrow$ CITY).

This attribute is decomposable into subattributes:

$A_1$ / ($\omega\tau \rightarrow$ (ZIP $\rightarrow$ CITY))
$A_2$ / ($\omega\tau \rightarrow$ (ZIP $\rightarrow$ (STREET $\rightarrow$ o))).

But there is a consistency constraint connected with A, namely "There is at most one ZIP code assigned to a given city and street" (types of variables obvious):

$CC_A = \lambda w\ \forall z_1\ z_2\ s\ c\ (([A_w\ (z_1,s)]=c \wedge [A_w\ (z_2,s)]=c) \supset (z_1 = z_2))$.

Decomposing the attribute A into $A_1$ and $A_2$, we get:

$A = \lambda w\ \lambda z s\ \iota c\ ([A_{1w}\ z] = c \wedge [[A_{2w}\ z]s])$,

and we have to transform $CC_A$ into $CC_{A1A2}$:

$CC_{A1A2} = \lambda w\ \forall z_1\ z_2\ c\ s\ (([A_{1w}\ z_1] = c \wedge [[A_{2w}\ z_1]s] \wedge [A_{1w}\ z_2] = c \wedge [[A_{2w}\ z_2]s]) \supset (z_1 = z_2))$.

Now we are ready to define the data kernel of a set of attributes (constructed by a database conceptual system describing the given part of reality, i.e., of a set of attributes of our interest).

**Definition 17**: *Data kernel* of a set of attributes *A* — defined in a given database conceptual system — is a set of such attributes *K* for which the following holds:

a) $K =_i A$

b) Members of *K* are undecomposable attributes

c) There are no two subsets $K_1$, $K_2$ of *K* ($K_1 \neq K_2$) such that $K_1 =_i K_2$. □

From the above considerations and statements it follows that a data kernel *K* of a set of attributes *A* is a minimum subset of *A* that consists of elementary attributes, and which is informationally equivalent with the set *A*. The following statement can be easily proved:

**Statement 6**: Let *K* be a kernel-like set of attributes, i.e. a set satisfying conditions (b), (c) of the above Definition 17, $K_1$, $K_2$ any two subsets of *K*. If $K_1 \leftarrow_D K_2$ then $K_1 \subset K_2$.

*Proof*: If $K_1$ were not a subset of $K_2$, then $K_1 \cup K_2 \neq K_2$, $K_1 \cup K_2 =_i K_2$ (Statement 2), which contradicts the condition (c) of the definition.

### *4.3. Lattice model of informational capability*

In Section 4.1. we showed that the connection of attributes with information consists in associating attributes with sets of propositions generated in a state-of-affairs *w*. Informational capability of attributes has been defined (Definition 8), and Statement 1 claims that comparing attributes as for their informational capability can be realised by the definability relation ($\leftarrow_D$). Following these results we would now like to order attribute sets according to their informational capability, i.e., by utilizing the $\leftarrow_D$ relation. But this relation is in general a *quasi-ordering* on attribute sets: it is reflexive and transitive, but it is *not antisymmetric*.

Consider, e.g., attribute sets {A}, {A, B}, such that B $\leftarrow_D$ A (A, B attributes). Then {A} $\leftarrow_D$ {A, B}, {A, B} $\leftarrow_D$ {A}, hence {A, B} $=_i$ {A} (cf. Statements 1, 2). However, {A} $\neq$ {A, B}.

*Example*: Set of attributes {A, B, C}, where
A = '(#STUDENT)-s of a (#COURSE)'
B = '(#COURSE) which is given in a (#ROOM) in an (HOUR)'
C = '(#STUDENT)-s who are lectured in a (#ROOM) in an (HOUR)'
(types of attributes A, B, C are obvious), is not a kernel-like set.

Either the attribute B or the attribute C is redundant: {B} $\leftarrow_D$ {A, C}, {C} $\leftarrow_D$ {A, B}. The respective constructions ensuring definability are as follows:

$B = \lambda w\, \lambda rh\, \iota c\, \exists s\, ([[A_w\, c]s] \wedge [[C_w\, (r\, h)]s])$

$C = \lambda w\, \lambda rh\, \lambda s\, \exists c\, ([[A_w\, c]s] \wedge [[B_w\, (r\, h)] = c])$.

Hence sets of attributes {A, B} and {A, C} are informationally equivalent: {A, B} $=_i$ {A, C}.

Yet in a special case of a kernel-like set $K$ of attributes a partial ordering on the power set $P(K)$ of the set $K$ can be easily defined. This partial ordering is based on the $\leftarrow_D$ relation, and due to the condition (c) of Definition 17 and Statement 6, respectively, it is the same as that one induced by the set-theoretical inclusion relation $\subseteq$. Hence the partial ordered set $L_K = (P(K), \leftarrow_D)$ is a complete lattice ([Birkhoff 1968], [Szasz 1963]), in which meet and join are defined as the set-theoretical intersection $\cap$ and union $\cup$, respectively. The least element of $L_K$ is an empty set {}, the greatest element is the set $K$.

This is rather a trivial result. In order to partially order attribute sets that are not necessarily kernel-like, we will utilise the method of a factor set induced by the equivalence relation $=_i$ ($\leftrightarrow_D$).

**Definition 18.** Let $\underline{A}$ be a set of attribute sets, $A \in \underline{A}$.

We define the *class generated by A* as $[A] = \{A' \mid A' \in \underline{A}, A' =_i A\}$, i.e. $[A]$ is the class of attribute sets informationally equivalent with $A$ (there can be infinitely many such attribute sets, of course).

The *factor set on* $\underline{A}$ with respect to the equivalence $=_i$ is defined as the set of all equivalence classes induced by $=_I$; hence $\underline{A}/=_i$ is the set $\{C \mid C \subset \underline{A}, C = [A]$ for some $A \in \underline{A}\}$.

Finally, *partial ordering on* $\underline{A}/=_i$ is defined as follows: $[A] \leq_i [B]$ iff $A \leftarrow_D B$ for any $A \in [A]$, $B \in [B]$. □

Though the $\leq_i$ relation is not an inclusion relation it corresponds to the partial ordering based on the $\Rightarrow$ relation (Definition 6) defined on the SIP classes of semiidentical propositions generated from particular attribute sets. In other words, there is an isomorphism between the factor set of attribute classes ordered by $\leq_i$ and the set of SIP classes ordered by $\Rightarrow$. As a result, $\leq_i$ is congruent with the informational capability of attribute sets.

Now having a lattice $L_K = (P(K), \leftarrow_D)$, we can extend it as follows: Let $K_1,\ldots,K_{2n}$ be members of $P(K)$. By substituting $[K_i]$ for $K_i$ ($1 \leq i \leq 2^n$) we obtain a partial ordered set of equivalence classes $L_1 = (\{[K_i]\}, \leq_i)$.

**Statement 7**: The partially ordered set $L_1 = (\{[K_i]\}, \leq_i)$ is a lattice in which meet ($\wedge$) and join ($\vee$) are defined as follows:

$$[K_i] \wedge [K_j] = [K_i \cap K_j]$$

$[K_i] \vee [K_j] = [K_i \cup K_j]$,

$K_i, K_j \in P(K)$, $1 \le i,j \le 2^n$, $K$ a kernel-like set of attributes.

*Proof*: We have to prove that for any $K_i$, $K_j \in P(K)$, $[K_i \cap K_j] = \inf \{[K_i], [K_j]\}$ and $[K_i \cup K_j] = \sup \{[K_i], [K_j]\}$. Obviously $[K_i \cap K_j] \le_i [K_i]$, $[K_i \cap K_j] \le_i [K_j]$. Let $[K_l]$ ($K_l \in P(K)$) be such a class that $[K_l] \le_i [K_i]$, $[K_l] \le_i [K_j]$. Then $K_l \subseteq K_i$, $K_l \subseteq K_j$ ($K$ is a kernel-like set (Statement 6) and, therefore, $K_l \subseteq K_i \cap K_j$. Hence $[K_l] \le_i [K_i \cap K_j]$. Analogously for the supremum.

The proof of the statement immediately follows also from the fact that the partially ordered sets $L_k$, $L_1$ are isomorphic with respect to the mapping $K_i \rightarrow [K_i]$ ($1 \le i \le 2^n$). Lattice $L_1$ is thus also a complete lattice of a finite length (though the members of particular $[K_i]$ can be infinite) with the least and greatest elements $[\{\}]$ and $[K]$, respectively.

*Note*: In this proof, and in all the following considerations, we naturally assume that if $A \subseteq B$ then $[A] \le_i [B]$, i.e., that our logic is monotonous. This is justified by the assumption of the 'correct data collection', the restrictions of which are specified by analytical consistency constraints.

Another lattice $L_2$ can be obtained as follows: Let again $K$ be a kernel-like set of attributes. We complement $K$ by all the attributes that are definable from $K$. Let the resulting set of attributes (which is, of course, no more kernel-like, and which can be infinite) be denoted by $A$, and the power set $P(A)$ by $\underline{A}$. The following Statement 8 shows that the factor set of equivalence classes $\underline{A}/=_i$ ordered by $\le_i$ is a lattice as well.

**Statement 8**. The partially ordered set $L_2 = (\underline{A}/=_i, \le_i$ is a lattice in which meet ($\underline{\wedge}$) and join ($\underline{\vee}$) are defined as follows:

$[A_i] \underline{\wedge} [A_j] = [ \cup_{k,l} (A_k \cap A_l) ]$, where $A_k$ is any element of $[A_i]$, $A_l$ is any element of $[A_j]$,

$[A_i] \underline{\vee} [A_j] = [ A_k \cup A_l ]$.

*Proof*: We have to prove that $[ \cup_{k,l} (A_k \cap A_l) ]$ is the infimum of $\{[A_i], [A_j]\}$, $[ A_k \cup A_l ]$ is the supremum of $\{[A_i], [A_j]\}$. Since the latter is obvious, we only prove the former. Obviously, $[ \cup_{k,l} (A_k \cap A_l) ] \le_i [A_i]$, $[ \cup_{k,l} (A_k \cap A_l) ] \le_i [A_j]$. Let $A' \in P(A)$ be any set such that $[A'] \le_i [A_i]$, $[A'] \le_i [A_j]$. Then any attribute $A \in A'$ is definable both from $A_i$ and $A_j$; thus there are sets $A_m \in [A_i]$, $A_n \in [A_j]$ such that $A \in A_m$, $A \in A_n$. Hence $A \in A_m \cap A_n$, and $A \in [\cup_{k,l} (A_k \cap A_l)]$, which means that $A' \subseteq [ \cup_{k,l} (A_k \cap A_l) ]$, $[A'] \le_i [ \cup_{k,l} (A_k \cap A_l) ]$.

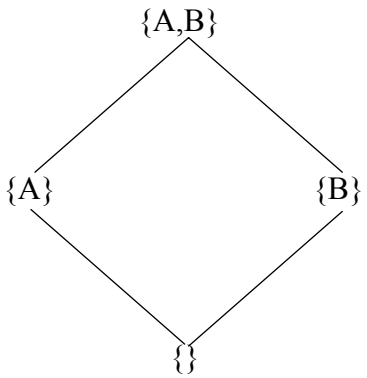
The least and greatest elements of $L_2$ are, again, $[\{\}]$ and $[K]$, respectively, $[K]$ being identical with $[A]$. This lattice can be of an infinite length: Consider, e.g., a subchain of $L_2$ $[\{A\}] \ge_i [\{A'\}] \ge_i [\{A''\}] \ge_i ....[\{\}]$, the length of which can be infinite in case of an infinite number of attributes A', A'', ... such that $A' \leftarrow_D A$, $A'' \leftarrow_D A'$, ... .

**Statement 9**. A sublattice $L_2'$ of $L_2$,
$L_2' = (\{[K_i]\}, \le_i, \underline{\wedge}, \underline{\vee})$ is isomorphic with the lattice
$L_1 = (\{[K_i]\}, \le_i, \wedge, \vee)$, where $K_i$ , $1 \le i \le 2^n$, are subsets of $K$.

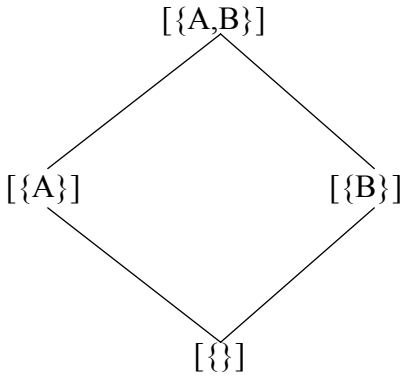
*Proof*: It is sufficient to prove that $[K_i] \underline{\wedge} [K_j] = [K_i] \wedge [K_j]$, i.e., that $[K_i \cap K_j] =$
$[ \cup_{k,l} (A_k \cap A_l) ]$, where $A_k$, $A_l$ are any elements of $[K_i]$, $[K_j]$, respectively. Obviously $[K_i \cap K_j] \le_i [ \cup_{k,l} (A_k \cap A_l) ]$. Let $A$ be any member of $\cup_{k,l} (A_k \cap A_l)$.
Then $A \leftarrow_D K_i$, $A \leftarrow_D K_j$. Since $K_i$, $K_j$ are kernel-like sets, it holds that $A \leftarrow_D K_i \cap K_j$.
Hence $[ \cup_{k,l} (A_k \cap A_l) ] \le_i [K_i \cap K_j]$, and, therefore, $[ \cup_{k,l} (A_k \cap A_l) ] =_i [K_i \cap K_j]$.

*Example*: Consider a kernel-like set of attributes {A, B}. Let A' $\leftarrow_D$ A, B' $\leftarrow_D$ B. Lattices $L_K$, $L_1$, $L_2$ can be illustrated by the following Hasse's diagrams:

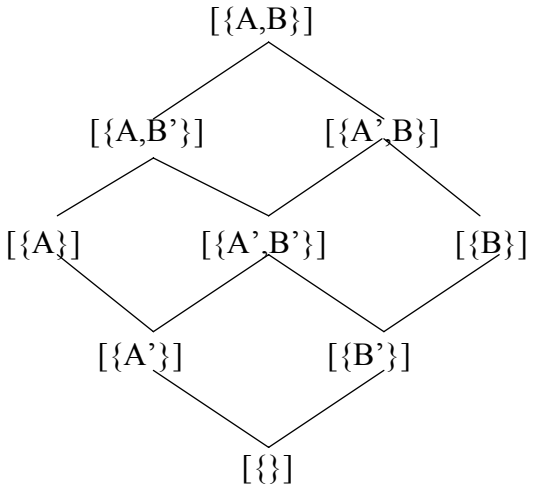
$L_K$




$L_1$




$L_2$




There are some practical consequences of the above described lattice theory of informational capability. Such an information lattice should be found in every real information system. There are three levels of a database application [Tsichr 1975]: an external one that includes a simplified model of the real world as seen by one or more applications;

conceptual, including a model of the world maintained for all applications of the enterprise; and internal, including a model of the data maintained for the computer representation of this limited model of the world. Though current modern database system architectures (for instance federative architecture, distributive architecture) do not usually realise particular schemata in a full accordance with this three level model (there are usually more than one schema in a particular level), the following principles should be followed, providing the system is correctly designed: (We will call attribute sets describing data in particular levels a conceptual set, an external set and an internal set.)

a) A conceptual set of attributes as well as an internal set should be members of the highest class, i.e. they must posses the greatest informational capability.

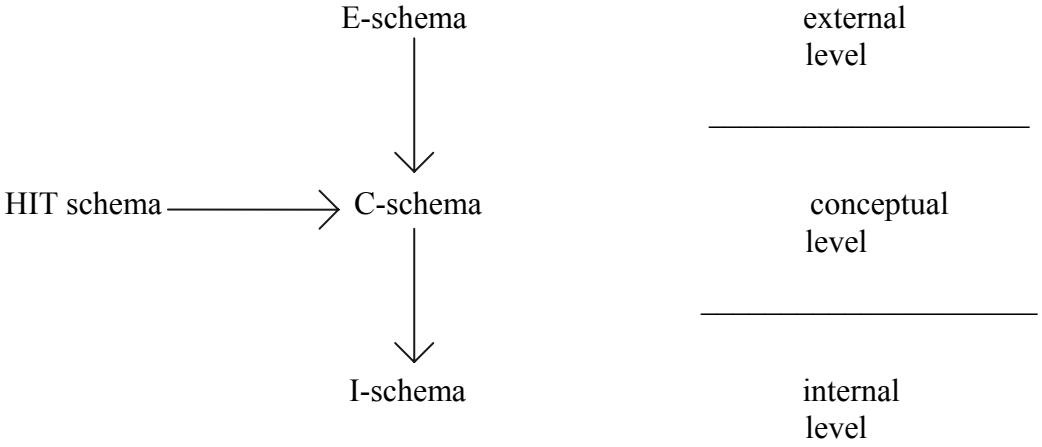
b) While the conceptual set of attributes — as an invariant of the system — should be a kernel-like set, the internal set may contain some redundant data, i.e. a controlled degree of redundancy of the stored data may be useful. Though reducing the degree of redundancy decreases demands on disc storage and facilitates updates, there may be some reasonable redundant data storage: This is the case when the function realising definability of attributes is difficult to implement with software and hardware at our disposal, and its performing is a lot of time or space consuming process, so that redundant data enhance efficiency of the system. The redundancy may also enhance the reliability of the system when using redundant information to reconstruct the database state that has been lost by a failure of the system.

c) Particular external sets of attributes are members of lower classes. The descending ordering of these classes mirrors the ascending ordering of particular management levels. Lowest classes contain highly aggregated information needed on higher management levels (top management). The classes on the same level are not comparable, for they describe different subsystems of a business application. Thus the lattice of equivalence classes of attribute sets illustrates the information flow in a correct information system.

# 5. Schema transformation, database design

## *5.1. Four-schema architecture*

The process of the information system design by means of the HIT method can be characterised by a thorough adhering the principle of the four-schema architecture [Staníček 1986], which is illustrated by the following Figure:




The roles of particular schemas can be characterized as follows:

- **HIT schema (conceptual)**

  serves for recording constructions of objects, their structure, characteristics and mutual relationships, i.e. for expressing concepts of objects of interest (UoD) and thus recording the semantics of natural language expressions that denote these objects; in other words, conceptual schema is an abstract model of the given part of reality. HIT schema defines informational demands that the system shall promote. At the same time it also serves as a common platform for users and designers of the information system, it is an invariant of the whole system. It is completely independent of the following way of implementation. It has to be comprehensive both to designers and users. We could see that the HIT conceptual schema (which is the result of the reality–mapping phase) defined as the couple of attribute and integrity-constraints constructions together with its graphical support meets these demands.

- **C-schema (central)**

  serves as a unifying schema of the whole system: it records the general data structures as they are described in the HIT schema but in a form easier to implement. Hence it is a computer-oriented representation of the HIT schema which is nevertheless still completely independent of a DBMS used to implement the system, and it makes it possible to implement the system in any DBMS though relational system is currently the most frequent one. In the HIT method it is represented by the *data type network* (DTN). C-schema is the result of the transformation HIT $\rightarrow$ DTN, and it basically corresponds to the Chen's E-R schema.

- **E-Schema (external)**

serves as a definition of user's views of data in the information system, and it is therefore a basic means of the user's communication with the information system. User-comprehensive descriptions of data structures are derived from the HIT schema; data structures to be manipulated by a communication means are derived from the C-schema. There may be a number of E-schemata in the information system. But it always has to be guaranteed that the informational capacity of the HIT schema (C-schema) is a summary of the informational capacities of particular E-schemata.

- **I-schema (internal)**

  is used for the implementation of the data base of the information system. It already determines the used programming techniques, namely the chosen DBMS, or at least the type of the DBMS. When the relational data model is used at this level (which is nowadays nearly exclusively valid), then the implementation in a relational DBMS will certainly follow. Network or hierarchical model can be used as well, but these we mention only for historical reasons. The role of an I-schema consists in optimally meeting the demands formulated in the HIT schema and represented in the data structures of the C-schema. Building up the I-schema is a subject of the information system design phase, and will be described in Section 5.3.

### *5.2. Transforming HIT conceptual schema into the C-schema*

At the beginning of Section 4.2 we mentioned the problem of polarity described in [Hull 1987], i.e. the problem of 'dual viewing' attributes: either as (n-ary) empirical functions or as complex (encapsulated) objects. We will now solve this problem using the formalism of TIL by means of a key notion of the transformation of a schema. The HIT data model enables us to work with objects and their attributes in a very natural way on the user's level of abstraction. Encapsulation is ensured by a schema on a different level of abstraction; the corresponding schema transformation is used to hide the internal structure of the objects being manipulated.

According to the HIT method of designing a database system the designer creates the HIT conceptual schema first, „nodes" of which are entity and descriptive sorts, and in which HIT attributes map the associations between these sorts. Following the principle of data independence according to which a conceptual schema should serve as an invariant of the system, we define the conceptual schema as containing a kernel of the database conceptual system. A set of HIT attributes describes reality in a user-friendly way but it is not easy to implement in a direct way. „User-friendliness" is achieved by the fact that HIT notion of attribute is rather stronger than the analogous one of other data models (cf. relational attribute, Chen's attribute, etc.). First, it covers not only descriptions of particular entities, such as name, identity card number, etc., but also links, associations between these entities (suppliers of a material,...). Moreover, HIT attribute is generally an $n$-ary ($n \geq 1$) function, which makes it possible to model functional dependencies between basic objects of interest in a natural way. Covering both descriptive attributes and associations by the HIT-attribute (compare with ORM's facts) makes the model extremely stable. Last but not least, this fact makes it possible to provide attributes with semantically exact names formulated in a natural-like language. But these $n$-ary functions are not easy to directly implement and thus most of the current database models fix these relationships as basic objects (of a more complex type) and allow only simpler unary descriptive attributes (cf. Chen's data model [Chen 1976]). The transformation of the HIT conceptual schema into such a flatter schema consists in 'binarisation' of $n$-ary attributes, i.e. the principle of representing $n$-ary functions by means of relationship sets is

applied, and particular consistency constraints are transformed. 'Binarisation principle' is described by the following statement:

**Statement 10.** A set of attributes $A_1$, $A_2$, ..., $A_k$ of types
$\omega\tau \rightarrow ((E_1,...,E_n) \rightarrow D_m)$ or
$\omega\tau \rightarrow ((E_1,...,E_n) \rightarrow (D_m \rightarrow o))$ ,
where $E_j$, j = 1,...,n, are entity sorts, $D_m$, m = 1,...,k, are descriptive sorts, is informationally equivalent with the set $\{B_1,...,B_n, B_{n+1},...,B_{n+k}\}$, where $B_i$ (i=1,...,n) are constructed as follows (remember that $r_{(i)}$ is the i-th projection of $r$):

$$B_i = \lambda w\, \lambda r\, \iota e_i\, (e_i = r_{(i)} \wedge (\exists d_1\, ([[A_{1w}\, r] * d_1]) \vee \exists d_2\, ([[A_{2w}\, r] * d_2]) \vee \ldots \exists d_k\, ([[A_{kw}\, r] * d_k])))$$

and $B_{n+m}$ (m=1,...,k) are constructed as follows:
$B_{n+m} = \lambda w\, \lambda r\, \lambda d_m\, ([[A_{mw}\, r] * d_m])$.
(Variables: $e_i/E_i$, $d_m/D_m$, $r/(E_1,...,E_n)$; symbol * stands either for the symbol = or it means application to $d_m$.)

Similarly an attribute A of a type
$(\omega\tau \rightarrow ((E_1,...,E_{n-1}) \rightarrow E_n))$ or $(\omega\tau \rightarrow ((E_1,...,E_{n-1}) \rightarrow (E_n \rightarrow o)))$ is informationally equivalent with the set of attributes $\{B_1,...,B_n\}$, where $B_i$ (i = 1,...,n) are constructed as follows:
$B_i = \lambda w\, \lambda r\, \iota e_i\, (e_i = r_{(i)} \wedge ([[A_w\, (r_{(1)},...,r_{(n-1)})] * r_{(n)}\, ]))$.

*Proof*: It is sufficient to prove that every $A_m$ (m = 1,...,k) is definable from $\{B_1,...,B_{n+k}\}$, and that A is definable from $\{B_1,...,B_n\}$. Indeed,
$A_m = \lambda w\, \lambda r\, \lambda d_m\, ([[B_{(n+m)w}\, r] d_m])$ or $A_m = \lambda w\, \lambda r\, \iota d_m\, ([B_{(n+m)w}\, r] = d_m)$, for any $m \in \{1, k\}$, and
$A = \lambda w\, \lambda r_{(1)} ... r_{(n-1)}\, \lambda r_{(n)}\, ([B_{iw}\, r] = r_{(i)}\, )$ or $A = \lambda w\, \lambda r_{(1)} ... r_{(n-1)}\, \iota r_{(n)}\, ([B_{iw}\, r] = r_{(i)}\, )$, for any $i \in \{1, n\}$.

Thus a new relationship set $R = (E_1,...,E_n)$ comes into being and the new attributes $B_i$ are unary. Populations of the sort $R$ consist of those tuples $(E_1,...,E_n)$ which are 'bound' by attributes $A_i$ or A, and the are constructed as follows:

$$[R_w] = \lambda e_1\, e_2\, ...\, e_n\, \exists w\, (\exists d_1\, ([[A_{1w}\, (e_1\, e_2\, ...\, e_n)] * d_1\, ]) \vee \exists d_2\, ([[A_{2w}\, (e_1\, e_2\, ...\, e_n)] * d_2\, ]) \vee \ldots \exists d_k\, ([[A_{kw}\, (e_1\, e_2\, ...\, e_n)] * d_k\, ]))$$

or in the case of attribute A

$$[R_w] = \lambda e_1\, e_2\, ...\, e_n\, \exists w\, ([[A_w\, (e_1\, ...\, e_{n-1})] * e_n\, ]).$$

Statement 10 makes it possible to transform a HIT conceptual schema into the so called *central schema* corresponding to Chen's Entity Relationship schema in such a way that the two schemas are equivalent in the following way:

**Definition 19.** Let $S = (A, C_A)$, $S' = (A', C_A')$ be database schemas. We shall say that $S$, $S'$ are *equivalent*, iff

a) The set of attributes of schema $S$ is informationally equivalent with the set of attributes of schema $S$'

b) Consistency statements constructed by $C_A$ and $C_A$' determine the same set of admissible states-of-affairs. □

*Central schema* $C = (B_N, C_B)$ obtained by the transformation of a HIT conceptual schema is such a schema in which attributes $B_1,...,B_n$ constructed by $B_N$ are 'unary', i.e., they are of a type $(\omega\tau \rightarrow (T_1 \rightarrow T_2))$ or $(\omega\tau \rightarrow (T_1 \rightarrow (T_2 \rightarrow o)))$, where $T_1$, $T_2$ are elements of the base $N$.

The method of transformation of the HIT conceptual schema $K = (A_S, C_A)$ over a base of sorts $S$ into an equivalent central schema $C = (B_N, C_B)$ over a base $N$ consists of the following three steps:

1. Enriching the base of sorts $S = E \cup D$ (where $E$ is the set of entity sorts, $D$ is the set of descriptive sorts) by the respective relationship sets $R$. Hence $N = E \cup D \cup R$.
2. Transforming $n$-ary ($n \geq 2$) attributes constructed by $A_S$ into unary attributes over the base $N$ according to the Statement 10.
3. Transforming consistency constraints $C_A$ connected with attributes $A_S$ into consistency constraints $C_B$ connected with attributes $B_N$ by substituting the respective constructions (cf. Statement 10)

   $\lambda r\, \lambda d_i\, [[B_{(n+i)w}\, r]\, d_i]$, $\lambda r_{(1)} ... r_{(n-1)}\, \lambda r_{(n)}\, ([B_{iw}\, r] = r_{(i)})$

   for $A_{iw}$, $A_w$, respectively, in $C_A$. The demand that $C_A$ and $C_B$ determine the same set of admissible states-of-affairs is obviously met due to the informational equivalence of the sets of attributes constructed by $A_S$ and $B_N$.

*Note:* The process of transformation can be fully automatized; it has been implemented by Duží in 1990 using the programming language Wander, see [BDSS 1990].

*Example*:
Consider a part of a HIT conceptual schema describing the purchase of a product. Supposing that one and the same product can be bought by different departments at different costs, we can have two undecomposable attributes:

$A_1$ – (DATE) when a given (#PRODUCT) has been bought for a given (#DEPARTMENT)

$A_2$ – (PRICE) that has been paid for a given (#PRODUCT) by a given (#DEPARTMENT)

Using Statement 10 we can transform $A_1$ and $A_2$ into four binary attributes of types:

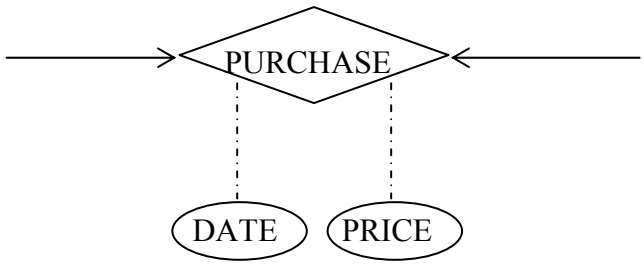
$B_1$ / (#PURCHASE) → (#PRODUCT)

$B_2$ / (#PURCHASE) → (#DEPARTMENT)

$B_3$ / (#PURCHASE) → (DATE)

$B_4$ / (#PURCHASE) → (PRICE)

The new relationship set (#PURCHASE) is a couple (#PRODUCT, #DEPARTMENT). The respective part of the C-schema can be illustrated by a variant of the E-R diagram:




Note that relations from entity sorts to relationship sorts are of the type 1:N (see the respective attributes $B_i$ – Statement 10).

In the central schema obtained in the above described way there are two kinds of attributes, namely *linkage attributes* and *descriptive attributes*. Linkage attributes are of a type

$$(\omega\tau \to (O_1 \to O_2)) \text{ or } (\omega\tau \to (O_1 \to (O_2 \to o))),$$

where $O_1$, $O_2$ are entity sorts or relationship sets, and descriptive attributes are of a type

$$(\omega\tau \to (O \to D)) \text{ or } (\omega\tau \to (O \to (D \to o))),$$

where $O$ is an entity sort or relationship set and $D$ is a descriptive sort.

A set of descriptive attributes describing one and the same entity or relationship sort is called a *data type*. Hence nodes of the central schema are data types, i.e. entity and/or relationship sorts with their relevant descriptions, links between these nodes are linkage attributes. Moreover, some descriptive sorts (or possibly tuples of descriptive sorts) are pointed out in the process of transformation, namely those descriptive sorts which occur in the type of more than one descriptive attribute or which appear as repeating items ($D \to o$) in a range of a descriptive attribute. We call them *marked descriptive sorts*, and they are mostly implemented in a similar way as data types. Considering each data type as a relation scheme in the above outlined way, we obtain (due to Statements 4 and 5) a schema in the 4th normal form (see 5.3.).

### *5.3. Transforming the C-schema into the I-schema (database design)*

Before briefly outlining the ways of possible I-schema design, we first describe the various normal forms used in („manual“) data schema normalisation. Afterwards we describe two basic ways of the I-schema design: either following some optimal normal forms principles or, in exactly defined and documented cases, breaking those principles. We also show that one of the benefits of using HIT method of database design is the fact that the resulting schema can be in an optimal normal form without ever wondering about any „post design“ manual normalisation techniques. This is due to the fact that using HIT method you express the UoD in terms of objects and their elementary functional dependencies.

Why a database designer is concerned with normalisation? The main reasons include [Becker 1999a]:

- Semantic grouping of related elements: Assigning ‘right’ attributes to the ‘right’ entities.
- Reduction of redundant values in tuples which can cause insert, delete and update anomalies.
- Avoiding the difficulties with inserting values to „non-existent“ entities.
- Reduction of null values in tuples.
- Disallow spurious tuples (incorrect join combinations of data values due to improper structures).

**Functional dependencies**

(Now we presuppose an acquaintance with the relational data model; using the term ‘attribute’ we mean a relational attribute.)

The main root of a normalisation technique centres on functional dependencies. This simply means that the values of a set of attributes Y in a relationship depend on, or are determined by, the value of a component X (determined $X \rightarrow Y$). It means that a functional dependency is invalid if we have two tuples with the same X value but different Y values.

For example, in many data structures, a ‘Social Security Number (SSN)’ determines a ‘Person’, which is commonly (but usually not uniquely) referenced by a ‘Name’. Thus we say that NAME is functionally dependent on SSN ($SSN \rightarrow NAME$). Keep in mind that the components X and Y may be composite structures (more than one relational attribute or, in other words, column).

Normalisation technique consists in ensuring that non-key attributes of a table are functionally dependent on the whole key of the same table.

**The normal forms**

A given data structure can be at one of several levels, or stages of completeness, of normalisation. These stages are known as normal forms. The eight normal forms are First Normal Form, Second Normal Form, Third Normal Form, Elementary Key Normal Form, Boyce-Codd Normal Form, Fourth Normal Form, Fifth Normal Form, and Project-Join Normal Form.

*First Normal Form*

First Normal Form (1NF) is now generally considered a part of the formal definition of a relation. Historically, 1NF was intended to disallow multi-valued attributes. 1NF dictates that the domains (allowable values) of attribute must include only atomic (simple, indivisible) values and that any given value of an instance of an attribute must be a single value from the domain of that attribute. In short, a given cell of a column in a table can contain only one value.

*Second Normal Form*

Second Normal Form (2NF) is based on the concept of a „full“ functional dependency. A functional dependency, $X \rightarrow Y$, is a full functional dependency if removal of one attribute from X means that the dependency does not hold any more. For example, given a table that tracks hours (HOURS) a given employee (SSN) devotes to a given project (PROJNUM), we note that HOURS is functionally dependent on the combination of SSN and PROJNUM

because a given employee can work on more than one project. Removal of either SSN or PROJNUM from the functional dependency results in an incorrect relationship.

A table is in 2NF if that table is in 1NF and every non-prime (is not involved in a primary key of the table) attribute in that table is fully functionally dependent on the primary key of the table.

For example, the following table is in 1NF but is not in 2NF because PNAME and PLOCATION are dependent on only part of the primary key (PROJNUM and SSN). Likewise, ENAME is also dependent on SSN.

Employee-Project table:

SSN, PROJNUM, HOURS, ENAME, PNAME, PLOCATION

To correct this schema, we need to create additional tables and decompose the partial dependencies into these new tables:

(Please note that, for simplicity, these tables will not denote the resulting referential integrity, such as foreign keys, that would need to be added to these decomposed schemas.)

Employee Table:

SSN, ENAME

Project Table:

PROJNUM, PNAME, PLOCATION

Hours Table:

SSN, PROJNUM, HOURS

*Third Normal Form*

Third Normal Form (3NF) is based on the concept of 'transitive dependency'. A transitive dependency can be loosely defined as a dependency that does not involve the primary key. For example, in the table below, we see that while all elements have functional dependencies on the key, SSN, there also exist other, transitive, dependencies. Namely, DEPTNAME is dependent on DEPTNUM.

Employee-Department Table:

SSN, ENAME, BDATE, DEPTNUM, DEPTNAME

A table is in 3NF if it is in 2NF and no non-key attributes are dependent on other non-key attributes.

We can decompose the above table into 3NF by creating a second table for department. Thus the following structure is in 3NF:

Employee Table:

SSN, ENAME, BDATE, DEPTNUM

Department Table:

DEPTNUM, DEPTNAME

There is a subtle difference between 2NF and 3NF. In 2NF, we are concerned about non-key fields being dependent on subsets of the key. In 3NF, we are concerned about non-key fields being dependent on other non-key fields. Another way to say this has been nicely summarised

as: any non-key field must be "...Dependent on the key, the whole key, and nothing but the key" [Kent 1983].

*Elementary Key Normal Form*

Elementary Key Normal Form (EKNM) is a subtle enhancement on 3NF (by definition, EKNF tables are also in 3NF) that most often occurs when there is more than one unique composite key (more than one column) which overlap (one or more columns are involved in both keys) in a table. Such cases can cause redundant information in the overlapping column(s). For example, in the following table, let's assume that a subject title (SUBJECTTITLE) is also a unique identifier for a given subject:

Enrollment Table:

| STUDENTNUM | SUBJECTCODE | SUBJECTTITLE |
|---|---|---|
| 1 | CS100 | ER |
| 1 | CS114 | ORM |
| 2 | CS114 | ORM |

This table, although it is in 3NF, violates EKNF. What is wrong with it? The primary key of the table is the combination of STUDENTNUM and SUBJECTCODE. However, we can also see a (non-primary) uniqueness constraint (alternate key) that should span the STUDENTNUM and SUBJECTTITLE attributes. The above schema could result in update and deletion anomalies because values of both SUBJECTCODE and SUBJECTTITLE tend to be repeated for a given subject. Decomposing the above table we obtain a schema satisfying EKNF:

Subject Table:

| SUBJECTCODE | SUBJECTTITLE |
|---|---|
| CS100 | ER |
| CS114 | ORM |

Enrollment Table:

| STUDENTNUM | SUBJECTCODE |
|---|---|
| 1 | CS100 |
| 1 | CS114 |
| 2 | CS114 |

For reasons that will become obvious in the following paragraph, ensuring that a table is in EKNF is usually skipped, as most designers will move directly on to Boyce-Codd Normal Form after ensuring that a schema is in 3NF. Thus, EKNF is included here only for reasons of historical accuracy and completeness.

*Boyce-Codd Normal Form*

Like EKNF, the only case a table is in 3NF but is not in Boyce-Codd Normal Form (BCNF) is when the table contains two or more candidate keys that overlap. Beyond that, there is only a subtle difference between EKNF and BCNF, which we outline now.

Consider the same example we used to illustrate EKNF, but we now add an attribute (GRADE) to denote a student's grade received in the course. (Further, for illustrative simplicity) let's assume that a student can only take a course once.) The following table violates BCNF:

Enrolment-Grade Table:

| STUDENTNUM | SUBJECTCODE | SUBJECTTITLE | GRADE |
|---|---|---|---|
| 1 | CS100 | ER | C |
| 1 | CS114 | ORM | A |
| 2 | CS114 | ORM | A |

We see that the GRADE attribute is now dependent only on a given enrolment pair and that the keys are now elementary (which satisfies EKNF). However, SUBJECTTITLE is dependent on SUBJECTCODE. Since the key of the table is STUDENTNUM and SUBJECTCODE, we decompose this structure into the following two tables which satisfy BCNF:

Course Table:

| SUBJECTCODE | SUBJECTTITLE |
|---|---|
| CS100 | ER |
| CS114 | ORM |

Enrolment-Grade table:

| STUDENTNUM | SUBJECTCODE | GRADE |
|---|---|---|
| 1 | CS100 | C |
| 1 | CS114 | A |
| 2 | CS114 | A |

One may note that this would also have happened to solve the EKNF problem in the previous section. For that very reason, most designers seldom worry about EKNF and move straight on to BCNF.

*Fourth Normal Form*

The final normal forms are concerned with multi-valued dependencies. We can also note that they are concerned with composite keys, as they tend to minimise the number of fields involved in a composite key.

A table is in Fourth Normal Form (4NF) if it is in BCNF and all functional dependencies are „single-valued". Another way to state this is to say that a table cannot contain two or more independent "multi-valued" facts [Kent 1983]. By independent we mean to say that there is no direct connection between the two (or more) multi-valued facts. This vague definition is

better handled by example. In the following table (in BCNF, since it is entirely composed of attributes involved in the key), we record people (NAME), instruments they play (INSTRUMENT), and music styles (MUSICSTYLE) they play:

| NAME | INSTRUMENT | MUSICSTYLE |
|---|---|---|
| Hallock | Piano | Classical |
| Hallock | French Horn | Classical |
| Hallock | Kazoo | Blues |
| Barden | Trumpet | Jazz |
| Hallock | Piano | Blues |

We see that redundancy occurs because a given person (NAME) can play more than one instrument (INSTRUMENT) and can play more than one music style (MUSICSTYLE) (the fact that Hallock plays piano is repeated, as is the fact that he plays the blues and classical). Further this table suggests a link between instruments and music styles. Can Hallock play blues with a French horn? Yes, he can.

In other words, we see that there are two independent multi-valued dependencies in the above table. The first is that a person (NAME) can play more than one instrument while the second is that a person (NAME) can play more than one music style. These facts are independent because these two facts have no bearing on each other. Decomposing this table into two tables in the 4NF solves the problem:

Plays Table

| NAME | INSTRUMENT |
|---|---|
| Hallock | Piano |
| Hallock | French Horn |
| Hallock | Kazoo |
| Barden | Trumpet |

Styles table:

| NAME | MUSICSTYLE |
|---|---|
| Hallock | Classical |
| Hallock | Blues |
| Barden | Jazz |

One should note that 4NF only applies to tables with three or more attributes (it eliminates overlapping multi-valued dependencies, which, by definition, require three or more attributes) and only when all attributes compose the primary key of the table.

*Fifth Normal Form and Project Join Normal Form*

Cases where table is in 4NF but is not in the Fifth Normal Form (5NF) are extremely rare. Further, Project Join Normal Form (PJNF) is a slightly stronger (although this is debated) case of 5NF, and in virtually all cases it can be treated as an equivalent. Therefore, PJNF is included here only for completeness.

As in 4NF, 5NF considerations apply only to tables with three or more attributes, all of which comprise the primary key.

The formal definition of 5NF PJNF requires using projection. A projection of a table is a subset of the total number of columns with no duplicate rows. For example, the following table can violate 5NF.

Trains table:

| EMPLOYEE | CLASSTYPE | COMPANY |
|---|---|---|
| Hallock | ORM | Visio |
| Hallock | UML | Visio |
| Hallock | ER | Visio |
| Hallock | Diagramming | Visio |
| Becker | ER | Oracle |
| Becker | ORM | Visio |
| Becker | UML | Visio |
| Becker | ER | Visio |
| Becker | Diagramming | Visio |

The table has the following projections:

Trains1:

| EMPLOYEE | CLASSTYPE |
|---|---|
| Hallock | ORM |
| Hallock | UML |
| Hallock | ER |
| Hallock | Diagramming |
| Becker | ER |
| Becker | ORM |
| Becker | UML |
| Becker | Diagramming |

Trains 2

| EMPLOYEE | COMPANY |
|---|---|
| Hallock | Visio |
| Becker | Oracle |
| Becker | Visio |

Trains 3

| CLASSTYPE | COMPANY |
|---|---|
| ORM | Visio |
| UML | Visio |
| ER | Visio |
| Diagramming | Visio |
| ER | Oracle |

The trains table may or may not be in 5NF depending on the business rules. Say we have to enforce the rule:

An EMPLOYEE trains a CLASSTYPE for a COMPANY if and only if an EMPLOYEE trains a CLASSTYPE, the EMPLOYEE trains for a COMPANY, and the COMPANY the EMPLOYEE trains for makes a tool that implements the CLASSTYPE the EMPLOYEE trains.

If we enforce this rule the Trains table is not in 5NF and must be reduced to the three tables represented by the above projections of the original table. To achieve 5NF, one checks all key tables for decompositions whose joins result in the same information. A cautionary note, however, is that such decompositions can lead to a loss of constraint knowledge. For example, in the above case, we need to create database code to handle the specified rule between an EMPLOYEE, the CLASSTYPES they train, and the COMPANY who makes the tool that implements the CLASSTYPE.

The root concept behind 4NF, 5NF and PJNF is that the tables not in these normal forms can be derived from simpler, more fundamental relationships. Further, 5NF does not differ from 4NF unless there are other rules (symmetric constraints) that dictate correct data population [Kent 1983]. Lastly, 5NF differs from 4NF in that the fact combinations we are concerned with are no longer independent from each other (due to the semantic constraints).

The previous discussion centres on what can be called „manual normalisation“. As you could see, this normalisation technique can be tricky and difficult to explain. Hence when designing a database schema by HIT method, we prefer data modelling technique that, in addition to many other benefits also happens to completely normalise the data structures with little extra “post-design” effort. For practical reasons we will be satisfied with the 4NF.

As it has been stated in the above Section 5.2, ‘nodes’ of the C-schema are data types, i.e. particular object classes together with the set of their descriptive attributes, linkage attributes describe relationships of a type 1:1, 1:M, M:N. Object classes are of two possible types: entity classes and relationship classes (aggregations, i.e. tuples of entity sorts). Now we

briefly outline the database design in the relational data model, i.e. transformation of the C-schema into the I-schema.

**Data types.**

Particular data types are realised as relational schemas, where the name of a schema is the name of the respective object class, descriptive attributes of the given class are relational attributes of the relational schema, the domains of which are particular representations of descriptive sorts.

**Keys and system identifications**

How shall we now realise relationships? The only way in the relational data model is realisation by means of *keys*, namely attributes ensuring the unique occurrence of a relation in every state-of-affairs. Keys are usually taken to be identification attributes. But there is a certain problem with this conception: Identification attributes are empirical functional dependencies and a completely correct data collection cannot be guaranteed, i.e. mistakes can occur when determining their values. Moreover, some object classes (especially the relationship classes) are identified by a *set* of attributes so that the respective key can be composite and rather long. Further, one and the same object can be identified in several different ways (for instance a person by its Social Security Number, or by its Birth-date Identity Number, or by a Personal Number in a given organisation, etc.). As a consequence we get the fact that one and the same object can be identified by different id-attributes in particular components of the system. How shall we then recognise that it is the same object? When taking into account the object-oriented approach to data design, then each real object should have in a database a unique image. Technical solving of this demand is a rather complicated problem. Modern object-oriented DBMSs provide us with the means for its realisation (they ensure the so-called *object-identity*). But nowadays there are still not many such systems, which could be actually called object-oriented, and if there are any in the market, they are rather expensive. Therefore it is recommended to introduce the so-called *system identification* of each object, i.e. a field uniquely identifying the object, which is calculated by the system.

**Realisation of relationships (linkage attributes)**

a) R*ealisation in 4NF*

Linkage attributes of the type 1:M are realised by repeating the key on the M-side. More precisely, a linkage attribute of a type $(O_1 \to (O_2 \to o))$ is realised by two relational schemas $O_1$, $O_2$, where the respective descriptive attributes of the objects $O_1$, $O_2$ are particular fields (relational attributes) of relations $O_1$, $O_2$. Relation $O_2$ will moreover contain a field—attribute which is the key of $O_1$. This situation is generally valid for such linkage attributes that are not 'partial on the 1-side' (object $O_2$ has a mandatory membership in the relationship), i.e. the number p in the ratio p,q:m,n of the attribute is equal to one. If it is not the case, then we have to realise such an attribute by three relational schemas, namely $O_1$, $O_2$ and moreover a schema R the attributes of which will be the id-attributes of the objects $O_1$, $O_2$. Hence such a relationship is realised analogically as the relationship M:N, see below. Note that the linkage attributes that we obtain using 'binarisation principle' have always the ratio 1,1:0,M, the new relationship set has an obligatory membership in the relationship.

*Note:* We did not pay an attention to 1:1 relationships, for they are just a special case of 1:M.

*Example*: Consider an attribute

(#ROOM) in which a given (#PACIENT) is placed / 0,1:0,M

When stating the ratio of this attribute we presupposed that, some rooms can be vacant, and, on the other hand, some patients do not have to be placed in any room, in other words, the system will record information also about non resident, out patients. This attribute will be realised be the following three relational schemas:

PACIENT(SSN, NAME, ADDRESS, ...)

ROOM(ROOM-NUMBER, LOCATION, NUMBER-OF-BEDS, ...)

PLACED(SSN, ROOM-NUMBER)

*Note*: This simple example illustrates the importance of distinguishing analytical vs. empirical constraints (see Section 3.2.4). Imagine that the user would insist on his being content with recording only resident patients, and would simply affirm that they do *not* have any non resident patients. In such a case we would record the above attribute as

(#ROOM) in which a given (#PACIENT) is placed / 1,1:0,M

with a note that the above total constraint is *empirical*. An admissible I-schema is now as follows:

PACIENT(SSN, NAME, ADDRESS, ..., ROOM-NUMBER)

ROOM(ROOM-NUMBER, LOCATION, NUMBER-OF-BEDS, ...)

*However*, such a schema would not be stable, unless we are content with null values (which should not be the case in the design phase), for the above constraint is only empirical, and the user can later easily change his mind and demand recording also non-resident patients. Hence in case of the empirical total constraint "on the 1-side" the former design (three relational schemes) is strictly recommended.

Relationships of the type M:N cannot be realised in a straight way. We have to use an analogical principle that we used when 'binarising' HIT attributes of the complexity greater than two. We introduce another relationship set and the original attribute is decomposed into two new attributes. For instance attribute

A = (#PERSON)-s who take part in an (#ACTION) / 1,M:0,N

is decomposed using a new relationship set, say

#PARTICIPATION = (#PERSON, #ACTION), and the attribute A is converted into two attributes $B_1$, $B_2$:

$B_1$ of the type (#PERSON $\rightarrow$ (#PARTICIPATION $\rightarrow$ o)) / 1,1:0,M

$B_2$ of the type (#ACTION $\rightarrow$ (#PARTICIPATION $\rightarrow$ o)) / 1,1:0,M

These two attributes are now realised as in the above case — relationships 1:M (the total constraint is now analytical!). As a result we obtain again three relational schemas PERSON, ACTION, PARTICIPATION.

There is also a problem with the realisation of multi-valued descriptive attributes. These attributes would violate 1NF. If a given DBMS does not make it possible to implement so-called 'nested relations', i.e. among others multi-valued attributes, then we have essentially two possibilities. An attribute A of the type (O $\rightarrow$ (D $\rightarrow$ o)) can be either realised as two relational schemas O and D that will be in the relation 1:M (hence we have to repeat the key

of the object O in the schema D), or as one relational schema in which there is one row for each value of D.

If we transform in the above described way the C-schema which originated in transformation of the HIT schema in which all the decomposable attributes have been decomposed, then the Statements 4 and 5 ensure that the resulting I-schema is in an optimal normal form (4NF in this case).

A special role in the schema is played by those *marked descriptive attributes* that have as its 'domain' a common descriptive sort. Hence the problem concerns such sets of attributes like $A_1 / (O_1 \rightarrow D)$, $A_2 / (O_2 \rightarrow D)$, ..., $A_n / (O_n \rightarrow D)$. As an example consider the descriptive sort ADDRESS. If there are more descriptive attributes with this sort in the C-schema, e.g.

(ADDRESS) of a given supplier (#ORGANISATION)

(ADDRESS) of a given customer (#ORGANISATION)

(ADDRESS) of a given contact (#PERSON)

(ADDRESS) where a given (#ACTION) is held

etc.,

and if by analysing user's demands during a dialogue we find out that the user will often put questions like „What is located on a given address?", then these facts signal that the expression 'address' denotes an entity sort rather than a descriptive one. In such a case we realise these attributes as if they were linkage attributes, i.e. the sort ADDRESS will result in a relational schema (which should be better called a LOCATION).

b) *Realisation violating 4NF ('nested relations')*

In some modern systems the 4NF is not strictly demanded. Normalisation yields the well known benefits like reducing redundancy and update anomalies (as described above), but sometimes it is so at the cost of effectiveness. As we have seen, keeping normal forms leads to introducing new relational schemas (new 'objects'), which means that in the run-time more join operations have to be performed, which are time demanding. Therefore some modern DBMSs make it possible to realise the so-called 'nested relations', especially multi-valued attributes, though at the same time the principle of non-redundancy is preserved.

In such systems we can then realise some M:N relationships and some multi-valued attributes without introducing another relation. Thus, for instance, we can decide to realise the following attribute

A = '(#PERSON)-s who take part in a given (#ACTION)' / 1,M:0,N

without additional relation PARTICIPATION in such a way that the couple PARTICIPATION is nested in the relation ACTION, i.e. relation ACTION will contain some attributes of participating persons. It, of course, has to contain the key, i.e., identification attribute(s) of participating persons. These attributes are multi-valued. Generally some relationship classes $R = (E_1,...,E_n)$ can be 'nested' into a relational schema corresponding to some of the entity sorts $E_i$. For instance, transforming HIT-attributes

$A_1$ = '(DATE) when a given (#PERSON) should be informed about a given (#ACTION)'

$A_1$ = '(HOUR) when a given (#PERSON) should be informed about a given (#ACTION)',

we obtain a new relationship class #INFORM = (#PERSON, #ACTION) with descriptive attributes 'DATE', 'HOUR' and linkage attributes

$B_1$ / INFORM $\rightarrow$ PERSON

$B_2$ / INFORM $\rightarrow$ACTION.

The corresponding relation INFORM can now be nested into, e.g., the relation PERSON which will then contain a set of multi-valued attributes:

- ACTION-CODE
- ACTION-DESCRIPTION
- DATE
- HOUR

Implementation of such queries as "About which actions shall we inform a given employee in the next week?" is then much more effective than in case the relationship class INFORM were realised as another relation (for it would demand performing a join operation of relations PERSON and INFORM). If, however, we expect frequent queries like "Which employees should be informed about a given action?", we can promote their effective realisation by means of the given DBMS (relational indices, etc.).

We have, however, to keep in mind that all such cases of breaking the principles of an optimal normal form have to be *carefully documented.* We have to know that the respective relationship classes did not disappear, but were nested into certain relations. Only in such a way we ensure that the database will not be „stiff" and a future redesign according to user's demands will be possible. If, for instance, a user wishes to record also a text of a notice that should be supported to the given employee together with the other information about an action, we have to know, to which relation the attribute 'text of a notice' should be added. Whether to PERSON or ACTION. In our above example this would mean to adjust the set of multi-valued attributes in the relation PERSON:

- ACTION-CODE
- ACTION-DESCRIPTION
- TEXT-OF-NOTICE
- DATE
- HOUR

*Note*: The Fourth Normal Form or Boyce-Codd Normal Form is sometimes not strictly demanded even in classical relational systems. The Third Normal Form is, however, recommended. In such a case we do not demand a careful applying decomposition statements. For instance, the relation ADDRESS = (STATE, TOWN, STREET, ZIP-CODE) would not be decomposed, for it is in the Third Normal Form (not Boyce-Codd Normal Form) though there is redundancy in storing the (name) of the city.

## 6. Conclusion

In this study we have dealt with the problems of conceptual data modelling. After a brief summary (which cannot be considered to be exhaustive, for such a task is completely out of the scope of the present study) of some well-known semantic (conceptual) data models we concentrated on the two models, namely COMIC and HIT which are both based on the theory of concepts. Whereas the COMIC system is based on the traditional Kaupi's theory of concepts, thus being rather intuitive and probably not sufficiently theoretically founded, the HIT data model stems from the Transparent Intensional Logic (TIL) and from the new non-traditional Materna's theory of concepts based on TIL. The core of this work can be characterised as a description of theoretical foundations of conceptual modelling using HIT data model. We thus concentrated on theoretical-logical aspects of the HIT data model, and rather neglected its practical consequences which have been described in [Duží 1986], [Duží 2000]. Of course, a good data model must prove its qualities in practice, and we can claim that the HIT data model has proved to be very intelligible for users, and has been successfully used when analysing and designing very large and complicated information systems. Nevertheless, the work of designers cannot be driven only by intuition, a good theoretical background becomes a great advantage. We have shown that all the currently used semantic data models actually do not essentially differ concerning modelling capabilities and constructs, but they all lack just a good theoretical background. Thus the used basic modelling notions are not precisely defined, the semantics is unclear, particular constraints are expressed in many different ways, most frequently in a natural language or in a programming language, the whole area seems to be rather „chaotic". We have shown that the HIT data model covers all the traditional modelling constructs, but unlike the above models it is well theoretically founded and all the modelling capabilities are handled in a uniform way.

The Entity-Relationship model is mostly considered to be quite an adequate model to business data analysis and has become nearly a standard nowadays. Why then do we advocate using HIT data model in the application analysis? A straightforward design of the central schema in the Entity-Relationship model is, of course, possible. But when the business reality is completely unknown to the designer or too complex, the design of relationship sets in the E-R model becomes a stumbling block. Moreover, using E-R, the designer has to determine particular entity sets, descriptive sets, attributes and relationships at the very beginning of the analysis. Any change of the conceptual view can thus cause redesigning the schema, which may sometimes be very burdensome. In such a case the designer can use the HIT method producing HIT conceptual schema first that is afterwards (automatically) transformed to the central schema, which produces particular entity and relationship sorts and their associations in a natural and at the same time exact way as the result of a careful analysis of functional dependencies between basic objects. Using the HIT method we can model *n*-ary (empirical) functional dependencies in a natural way as *n*-ary (empirical) functions, and we are not forced to fix these objects of a more complicated type as basic 'nodes' of a schema at the very beginning of the design process. Moreover, similarly as in some progressive, user-friendly and stable data models (e.g. ORM), using the HIT data model we do not distinguish between (descriptive) attributes and relationships between entity sorts, modelling them in a uniform way as *n*-ary HIT attributes, which makes the model stable and easy to use. Last but not least, by recording HIT attributes we obtain a precise documentation of the database conceptual schema including the semantics of particular objects and their relations for a future re-design.

Of course, the HIT data model, though being successfully used in practice, is still being developed. Further research in this area will concentrate on the HIT analysis of metadata and its usage for integrating information sources, studying metadata in the context of a Global Information System (GIS), and the development of a theoretical framework for the specification, querying and maintenance of GIS. We will aim at finding and realising possibilities of the support of GIS *cooperativity* and *interoperatibility*. Proposed models and procedures will be tested by a prototype software application that can be used as a tool for the work with metadata.

## References


[Abiteboul 1987] S. Abiteboul, R. Hull: IFO: A formal semantic database model, *ACM Trans. On database systems.* Vol.12, No.4, 1987, pp. 525-565.

[Alagic 1999] C. Alagic: A Family of the ODMG Object Models. Eder et. al. (Eds.): *ADBIS'99,* LNCS 1691, Springer-Verlag Berlin Heidelberg, 1999, pp. 14-30.

[Barendregt 1981] H.P.Barendregt: *The Lambda-Calculus*. North Holland, 1981.

[Becker 1999a] S.A. Becker: Data Schema Normalisation. *The Journal of Conceptual Modeling*, InConcept, Inc., June 1999.

[Becker 1999b] S.A. Becker: An Argument for the Use of ER Modeling. *The Journal of Conceptual Modeling*, InConcept, Inc., August 1999.

[Becker 1998a] S.A. Becker: Common Model Fragments: People and Organizations. *The Journal of Conceptual Modeling*, InConcept, Inc., April 1998.

[Becker 1998b] S.A. Becker: Normalization and ORM. *The Journal of Conceptual Modeling*, InConcept, Inc., August 1998.

[Becker 1998c] S.A. Becker: Building a Better Data Model. *The Journal of Conceptual Modeling*, InConcept, Inc., December 1998.

[Beeri 1978] C.Beeri, P.A.Bernstein, N.Goodman: A sophisticate's introduction to database normalization theory. *Proc. VLDB 1978*, pp. 113-124.

[Bell 1988] D.A.Bell: Efficient and flexible database query processing. *Proc.DATASEM'88*, ČSVTS Praha, pp. 209-225.

[BDSS 1990] M. Benešovský, M. Duží, P. Sojka, M. Šmídek: WANDER, a graph handling tool. In: Proc. 13th *ISDBMS*, Mamaia (Romania), 1990, pp. 110-121.

[Birkhoff 1968] G. Birkhoff, S. Mac Lane: *Algebra*. The Macmillan Company, New York, 1968.

[Bolzano 1837] B. Bolzano: *Wissenschaftslehre* I, II, Sulzbach, 1837.

[Booch 1999] G. Booch, J. Rumbaugh, I. Jacobson: *The Unified Modeling Language User Guide.* Addison Wesley 1999.

[Carnap 1952] R.Carnap, Y.Bar-Hillel: *An Outline of the Theory of Semantic Information*. Technical Report No. 247, MIT Research Laboratory in Electronics, 1952.

[Cattell 1997] R.G.G. Cattell, D. Barry, D. Bartels, M. Berler, J. Eastman, S. Gamerman, D. Jordan, A.Springer, H. Strickland, D. Wade: *The Object-Oriented Database Standard: ODMG-2.0*, Morgan Kaufman, San Francisco, CA, 1997.

[Chen 1976] P.P.S. Chen: The Entity-Relationship Model: towards a Unified View of Data. *ACM Trans. On Database Systems,* 1,1,1976, pp. 56-68.

[Church 1940] A. Church: A Formulation of the Simple Theory of Types. *Journal of Symbolic Logic*, 5,1,1940, pp. 337-387.

[Codd 1970] E.F. Codd: A Relational Data Model for Large Shared Data banks. *Comm. ACM,* 13,6,1970, pp. 377-387.

[Codd 1979] E.F. Codd: Extending the database relational model to capture more meaning. *ACM Trans. Database Syst.* 4,4,1979, pp. 397-434.

[Cresswell 1985] J.M. Cresswell: *Structured meanings*. MIT Press, Cambridge, Mass.

[Duží 1986] M. Duží, F. Krejčí, P. Materna, Z. Staníček: *HIT Method of the Database Design.* Research report, Technical University of Brno, Prague 1986.

[Duží 1988] M. Duží, P. Materna: Informational capability and distinguishing force of database attributes. *Database Technology*, Pergamon Press, Vol. 2, 2, 1988, pp. 4-9.

[Duží 1990] M. Duží, P. Materna: Attributes: Distinguishing capability versus Informational Capability. *Computers and Artificial Intelligence,* Vol. 9, 2, 1990, pp. 169-185.

[Duží 1991] M. Duží: Logic & Data Semantics. Thesis. Charles University of Prague.

[Duží 1992] M. Duží: Semantic Information Connected with Data. Proc. *Database Theory - ICDT'92,* Lecture Notes in Computer Science, Springer-Verlag, Berlin, pp. 376-390.

[Duží 1994] M.Duží, P. Materna: Non reasonable sentences. *Proc. Logica'94,* T.Childers, O.Majer (eds.), Czech Academy of Sciences, Prague, pp. 107-124.

[Duží 1996] M. Duží: Propositional attitudes and synonymous expressions. *Proc. Logica'96,* T.Childers et al (eds.), Czech Academy of Sciences, Prague, pp. 309-321.

[Duží 1997] M. Duží, J. Pokorný: Semantics of general data structures. *Information Modelling and Knowledge bases* IX, P.-J.Charrel et al (Eds), IOS Press 1998, pp.115-130.

[Duží 1999] M. Duží: Conceptual data modelling using transparent intensional logic. *Studies in Logic and Philosophy,* Vol. 4, Varieties of Conceptual Representation, Academy of Sciences of the Czech Republic, 1999, pp.13-34.

[Duží 2000] M. Duží: Logical Foundations of Conceptual Modelling using HIT Data Model. Proc. of the *10th European-Japanese Conference on Information Modelling and Knowledge Bases.* Ed. by H. Jaakkola, H. Kangassalo, Tampere University of Technology, Finland, 2000, pp. 161-176.

[EDV 1983] EDV Studio Ploenzke: *Integrated Software-Technologie.* Wiesbaden 1983-1991.

[Elmasri 1989] R. Elmasri, S. Navath: *Fundamentals of Database Systems.* Redwood City, CA: Benajmin/Cummings, 1989.

[Fowler 1997] M. Fowler, K. Scott: *UML distilled.* Addison-Wesley, 1997.

[Halpin 1998] T. Halpin: UML Data Models from an ORM Perspective. *The Journal of Conceptual Modeling*, InConcept, Inc., April 1998 - August 1999.

[Halpin 1999] T. Halpin: Entity Relationship Modeling from an ORM Perspective. *The Journal of Conceptual Modeling*, InConcept, Inc., December 1999.

[Hammer 1981] M. Hammer, D. McLeod: Database description with SDM: A Semantic database model. *ACM Trans. Database systems,* Vol.6, 1981, pp.351-386.

[Howe 1989] D.R. Howe: *Data Analysis for Database design.* 2nd Ed., Edw.Arnold, a division of Holder&Stoughton, London, 1989.

[Hull 1984] R. Hull, Ch.K.Yap: The Format Model: A theory of database organization. *Journal of the Association for Comp. Machinery*, 31,3,1984, pp. 518-537.

[Hull 1986] R. Hull: Relative information capacity of simple relational database schemata. *SIAM J. Comput*, 15,3,1986, pp. 856-886.

[Hull 1987] R. Hull, R. King: Semantic database modelling: Survey, applications and research issues. *ACM Computing Surveys,* Vol.13, 1987, pp.202-260.

[Hull 1991] R. Hull, M. Yoshikawa: On the equivalence of database restructurings involving object identifiers. *Proc. PODS'91*, 1991.

[ISO 1982] *Concepts and Terminology for the Conceptual Schema and the Information Base*, J. van Griethuysen ed., ISO/TC97/SC5/WG3-N695 Report, ANSI, New York.

[Junkkari 1999] M. Junkkari: Set theoretical specification for representing relationships among components,. *Studies in Logic and Philosophy* 4, Academy of Sciences of the Czech republic, Prague 1999, pp. 45-62.

[Kauppi 1967] R. Kauppi: *Einfuhrung in die Theorie der Begriffsysteme.* Acta Universitatis Tamperensis, Ser. A, Vol. 15, Tampere 1967.

[Kangassalo 1993] H. Kangassalo: COMIC: A system and methodology for conceptual modelling and information construction. *Data and Knowledge Engineering* 9 (1992, 1993), North-Holland 1993, pp. 287-319.

[Kangassalo 1998] H. Kangassalo: Conceptual Description for Information Modelling Based on Intensional Containment Relation. University of Tampere, 1998.

[Kent 1983] W. Kent: A Simple Guide to Five Normal Forms in Relational Database Theory. *Communications of the ACM,* 26(2), February 1983, pp. 120-125.

[King 1985] R. King, D. McLeod: A Database design methodology and tool for information systems. *ACM Trans. on Office Information Systems*, Vol. 3, 1985, pp. 2-21.

[Klug 1980] A. Klug: Entity-relationship views over uninterpreted enterprise schemas. *Entity-Relationship Approach to Systems Analysis and Design*, P.P.Chen, ed., North-Holland, Amsterodam, 1980, pp. 52-72.

[Lien 1980] Y.E. Lien: On the semantics of the entity-relationship model. *Entity-Relationship Approach to Systems Analysis and Design,* P.P.Chen, ed., North-Holland, Amsterodam, 1980, pp. 131-146.

[Ling 1981] T.W. Ling, F.W. Tompa, T. Kameda: An improved third normal form for relational databases. *ACM Trans. Database Systems*, Vol. 6, 1981, pp. 329-346.

[Maier 1980] D. Maier, A.O. Mendelzon, F. Sadri, J.D. Ullman: Adequacy of decomposition of relational databases. *Journal Comp. Systems Sci.*, Vol.21, 1980, pp. 368-379.

[Materna 1987] P. Materna: Entity Sorts: What are They? *Computers and Artificial Intelligence*, 6,4,1987, pp.321-324.

[Materna 1998] P. Materna: *Concepts and Objects*. Acta Philosophica Fennica, Helsinki, 1998.

[Materna 1999] P. Materna: Two Notions of Concept. *Studies in Logic and Philosophy* 4, Academy of Sciences of the Czech republic, Prague 1999, pp. 85-112.

[Mylopoulos 1978] J. Mylopoulos, P.A. Bernstein, H.K.T. Wong: A language facility for designing interactive database-intensive applications. *Proc. ACM Sigmod Int. Conf. Management of data,* Austin, Tex., 1978.

[Niemi 1999] T. Niemi: Algebraic definition for Kauppi's concept system and intensional queries in it. *Studies in Logic and Philosophy 4*, Academy of Sciences, Prague, 1999, pp. 113-124.

[Niemi 1998] T. Niemi: New Approaches to Intensional Concept Theory. Manuscript, University of Tampere, 1998.

[Niinimäki 1999] M. Junkkari, M. Niinimäki: An algebraic approach to Kauppi's concept theory. *Information Modelling and Knowledge Bases X*, H.Jaakkola et al. (eds.), IOS Press, 1999.

[Nilsson 1998] J.F. Nilsson, J. Palomäki: Towards computing with extensions and intensions of concepts. *Information Modelling and Knowledge bases* IX, P.-J.Charrel et al (Eds), IOS Press 1998, pp. 101-114.

[Palomäki 1994] J. Palomäki: *From Concepts to Concept Theory*. Discoveries, Connections and Results. Acta Universitatis tamperensis, Ser. A, Vol. 416, Tampere 1994.

[Palomäki 1997] J. Palomäki: Three Kinds of Containment Relations of Concepts. *Information Modelling and Knowledge Bases* VIII, H.Kangassalo et al. (Eds.), IOS Press, 1997, pp. 261-277.

[Pokorný 1998] J. Pokorný, I. Halaška: *Databázové systémy* (in Czech). Skripta ČVUT, Prague, 1998.

[Pokorný 1992] J. Pokorný: *Databázové systémy a jejich použití v informačních systémech* (in Czech). ACADEMIA Praha, 1992.

[Rumbaugh 1999] J. Rumbaugh, I. Jacobson, G. Booch: *The Unified Modeling Language Reference Manual.* Addison-Wesley 1999.

[Schewe 2000] K.D.Schewe: UML: A Modern Dinosaurus? - A Critical Analysis of the Unified Modelling Language. Proc. of the *10th European-Japanese Conference on Information Modelling and Knowledge Bases.* Ed. by H. Jaakkola, H. Kangassalo, Tampere University of Technology, Finland, 2000, pp. 188-207.

[Scholl 1990] M.H. Scholl, H.J. Schek: A relational object Model. Proc. *ICDT'90*, Paris, 1990.

[Sharp 1999] J.K. Sharp: Precise Meaning of Object Oriented Models. *The Journal of Conceptual Modeling*, InConcept, Inc., April 1999.

[Shipman 1981] D. Shipman: The functional data model and the data language DAPLEX. *ACM Trans. Database systems,* 6,1, 1981, pp. 140-173.

[Smith 1979] J.M. Smith, D.C.P. Smith: A database approach to software specification. Tech. Rep. CCA-79-17, Computer Corporation of America, Cambridge, Mass., 1979.

[Staníček 1986] Z. Staníček: Four-schema architecture of Information System. Lecture on 4th joint seminar on *DBMS*, Linkoping University, Sweden, May 1986.

[Storey 1988] V.C. Storey: *View Creation*. An Expert System for Database design. ICIT Press, Washington, 1988.

[Su 1979] S.Y.W. Su, D.H. Lo: A semantic association model for conceptual database design. *Proc. Int. Conf. Entity-Relationship Approach to Systems Analysis and Design.* Los Angeles, Calif., 1979.

[Szasz 1963] G. Szasz: *Introduction to Lattice Theory*. Akademiai Kiado, Budapest 1963.

[Tichý 1988] P. Tichý: *The Foundations of Frege's Logic.* De Gruyter, Berlin - New York, 1988.

[Tsichr 1975] D. Tsichritzis, A. Klug (eds): The ANSI/X3/SPARC DBMS Framework. Report of Study Group on Database Management Systems. *Information Systems*, 3,1, 1975, pp. 173-191.

[Ullman 1988] J.D. Ullman: *Principles of Database and Knowledge-base Systems.* Computer Science Press, Inc., 1988.

[Vaníček 1988] J. Vaníček: Information Aspects. A tool for redundancy control data. *Kybernetika*, 24, 4, 1988, pp. 293-306.

[Vianu 1987] V. Vianu: Dynamic Functional Dependences and Database Aging. *Journal of the Association for Comp. Machinery*, 34,1, 1987, pp. 28-59.

[Yao 1982] S.B. Yao, V. Waddle, B.C. Housel: View Modelling and integration using the functional data model. *IEEE Trans. Soft. Eng.*, SE-8, 6, 1982, pp. 533-544.

[Zlatuška 1986] J. Zlatuška: Data Bases and the Lambda Calculus. *Proc. IFIP'86 World Computer Congress*, Dublin, 1986, pp. 97-104.

[Zlatuška 1990] J. Zlatuška: Modelling Inheritance in a Strongly Typed Functional Model. *Workshop Kiev*, 1990.

[Zlatuška 1993] J. Zlatuška: *Lambda-kalkul.* Masarykova universita Brno, 1993.