On the Relationship between Information and Data

Gernot Richter, Gesellschaft fuer Mathematik und Datenverarbeitung (GMD), St. Augustin

Summary

On the background of a general model of information systems a view is analyzed which explicitly distinguishes between information and its representation. Using a conceptual system (IMC) which has been designed to talk about information structures and their manipulation, some ideas on the representation of information are presented. The significant role of type declarations for the information and representation level is shown. For the concepts of format and data a definition is outlined. In the light of these considerations some topics concerning present data base technology are discussed. This gives the motivation to conclude with a plea for conceptual differentiation in the field of data base management systems.

1. A model view of information systems

For information systems a view has been proven to be very useful which considers them consisting of communicating functional units (Funktionseinheiten) in the sense of [DIN]. Recently these functional units have been identified in [ANSI] as roles or work stations characterized only by their function within the system rather than by their technical realization. Years ago this kind of functional units has already been introduced and applied in [ABN] following a suggestion of C. A. Petri. There the term office (Instanz) has been chosen for the functional units under consideration. In information systems the offices influence each other by communicating messages. So the need has been recognized to introduce a complementary functional unit which allows for the exchange of messages between offices. To this kind of

functional units the term channel (Kanal) was given in [ABN]. The concept of interface, as used in [ANSI] has a direct relation to the concept of channel: An interface is a system of rules which govern the communication via a considered channel. Also a channel is characterized only by its function within the system serving as a facility where messages can be posted and taken by the communicating offices.

This yields a model view of information systems which provides for the decomposition into two distinct classes of functional units:
- offices characterized by the processes they can perform
- channels characterized by the states they can assume.

This model view applied to data base management systems recently has gained some publicity, since the publication of [ANSI] is under discussion both in the world of scientific research (IFIP/TC-2 and IAG) and in the area of standardization (ISO/TC 97/SC 5).

With the above model in mind we want to do a close look to the communication of two offices via one channel. This seems to be an adequate minimum configuration to examine the interrelation between information and data.

To illustrate this configuration we use the graphic notation of [PET], where offices are depicted by boxes and channels by circles (in the cited paper only elementary offices and channels are considered). This yields fig. 1. In the adopted model communication between both offices is done by exchanging messages via the linking channel. The arrows in the above figure only indicate the possibility of access and are n o functional units.

A further aspect is depicted in fig. 1: The exchange of messages makes only sense if both communicating offices have a common background of understanding, which allows them to interpret the messages found in the channel. The assumption of such a "universe of discourse" is a very useful auxiliary model for the understanding of communication also between technical functional units.

2. Model information and abstraction

So far no reference has been made to a distinction between information and data. But words as "represent" and "interpret" indicate a kind of mapping between two things. It is the goal of this section to show that there are two mappings to be considered. Both have the nature of an abstraction, i.e. omission of features not to be considered - but they start at different points.

One kind of abstraction starts with the so-called initial information (Ausgangsinformation), which is to be understood as the whole of knowledge or ideas a person has about something (of the real world or anything else). For a certain pragmatic context, i.e. pursuing an intended purpose it might be that not the whole information is needed but only the "relevant" part of it. The information about a person e.g. is different for administrative purposes and for medical purposes; the information about a technical process for teaching purposes will be different from what is needed for engineering purposes. So it is the intended purpose which controls the abstraction process. In [DURI] the result of the abstraction process has been called the (respective) model information (Modellinformation). In similar considerations of [STEEL] the above abstraction is called the "engineering abstraction" which yields the "engineering model". The term model information indicates, that we are still on the information level. In the present context we do not adopt any definition of information; the concept is used in the sense of knowledge or idea (about something). Thus information is viewed as being of mental nature.

It is obvious, that depending on the respective intended purpose various abstractions can be performed on the same initial information.

It is not of interest in this presentation, whether the model information "exists" or not - whatever that means. However we found the approach very useful which assumes a level of model information (as did also other authors).

Model information cannot be communicated directly because of its mental nature. There must be a representation of it (on a medium) which can be handed out to the addressee (or which can be stored for later use). Such a representation is what usually is called "data". The distinction between information and its representation is the background on which all the following ideas have been developed.

Now it is possible to show the other abstraction mentioned above, which is of a quite different nature. Consider some messages (here in the sense of data) which by agreement between the communicating offices have the same meaning. What is "same meaning" in the present case? Any message is considered to be a representation of model information. As already pointed out, exchange of messages is assumed to have the goal to exchange model information. There are rules for the mapping of messages to model information. Such a mapping usually is called "semantics" and the process of mapping "interpretation". If several messages are mapped onto the same model information, they all have the "same meaning". So we have an abstraction from various representations to the pertinent model information by ignoring the respective representational peculiarities.

There is one problem which might have been apparent already in the above discussion. Considering the communication beween an author and the audience he has the need of representing model information, which he wants to write about. For this purpose a kind of (graphical) reference language is beneficial, in which information can be represented and the interpretation of which is agreed upon. Such a graphical language will be presented in the following and used for canonical representation whenever emphasis is laid on the model information rather than on one of its possible representations.

3. Outlines of a conceptual model of information

Before dealing with any problems of representation the properties of model information itself have to be identified. What is an adequate view of model information with respect to applications? This question brings us into a (at least in the past) very controversal area of argumentation about the advantages and deficiencies of so-called "data models" (hierarchic, network, relational, ...). For general considerations we can avoid this topic by adopting a view which covers the various "data models". This view has been outlined in [DURI] and is reflected in a conceptual system called Information Management Concepts (IMC). These concepts have been developed as a means for talking about model information, in particular in the context of data base management systems. Simultaneously, rules for graphic representation of model information in terms of IMC were developed. Both the basic concepts of IMC and the related canonical representations will be outlined in this section to facilitate the treatment of the topic of "data" (in the

sense of representation) and its relationship to information.

In IMC any portion of model information which can be referred to in a communication is called a construct (Gebilde). A construct may be the information about a family, a car in an administration, a book in a library, a process in a factory. A construct is either an atom (Atom) or an aggregate (Aggregat). Whereas an atom is declared to "be", i.e. to be viewed as elementary (in a given situation), an aggregate is a compound construct, the composition of which is relevant in a considered communication. A construct in its capacity as a part of another construct is a component (Komponente). A construct cannot be a component within itself.

Depending on the way of immediate composition (first level) an aggregate is either a collection (Kollektion) or a nomination (Nomination). These two generic types of aggregates differ in that a collection is an unordered finite set of constructs, whereas a nomination is a (mathematical) function from names (Name) to constructs. The domain of a nomination therefore is a set of names. For the property of being a collection or a nomination the nature of the immediate components is of no significance. Names only serve for the selection of immediate components in a nomination (in the same manner as selectors in the Vienna Definition Language, cf. e.g. [ZEM]). Beyond that no meaning of names is involved within the framework of IMC.

To show examples of atoms, collections, and nominations we first have to introduce the above mentioned canonical representation. In IMC a box represents a construct. The composition of a construct is shown either by nested boxes (fig. 2) or by trees (fig. 3). In a tree representation the aggregation of constructs to an aggregate is expressed by the vertex. A combination of both representation techniques is possible. Atoms are always represented by boxes. In the representation of nominations the presence of names is depicted by small circles attached to the component representations. The names are written close to the circles. A detailed example of a "relation" and the corresponding "set network" in IMC representation is given in [DKR].

If we look at the representation of the nomination of fig. 2 or 3 we notice that the same construct may appear in different contexts. In a representation we can point at the various locations where (the representation of) the same construct appears, on the conceptual level of model information we cannot. Therefore a concept is needed which

allows to distinguish between different appearances of one construct (within a considered embracing construct). In IMC the concept of spot (Stelle) has been introduced. A spot can be defined as a sequence of pairs (name, construct). In case of a collection the empty name is inserted at the name position in the pair. The first pair of a spot defining sequence always consists of the empty name and the reference construct, in (=relative to) which the spot is considered. So with the symbols of fig. 4 the construct in question appears at the spots

$(-,c_1)$ (home address,c_2) (city,c_3)
$(-,c_1)$ (place of birth,c_3)
$(-,c_1)$ (branches,c_5) $(-,c_3)$

which are spots in c_1. (The lower case c's stand for the respective construct.) The same construct also appears at the spot

$(-,c_2)$ (city,c_3) in c_2 and
$(-,c_5)$ $(-,c_3)$ in c_5.

Another example is c_7 which appears in c_1 at the following two spots:

$(-,c_1)$ (home address,c_2) (street,c_6) (number,c_7)
$(-,c_1)$ (date of birth,c_4) (year,c_7)

It turns out, that the concept of spot is essential for the discussion and understanding of some sophisticated aspects in data base management systems, not least those concerning the interrelationship between information (constructs) and data (representations).

Fig. 2 and 3 show, by the way, that in canonical graphic representation always constructs a t s p o t s are depicted. As by definition any spot structure is hierarchic, one might be tempted to label IMC a hierarchic system. But it is obvious, that in a l l existing information models (in hierarchic, network, relations, etc.) the spots form hierarchic trees.

So far only individual constructs have been considered. Nothing about types or declarations has been said nor used tacitly. A type in general is a set. But not any set is a type. First of all, it has to be determined what are the elements of such a set. In the present context we focus on types of constructs (Gebildetyp), thus the elements are constructs. In the world of data base management systems instead of

"element" the terms "occurrence" or "instance" of a type have been adopted.

But not even any set of constructs is a construct type. A type of constructs has to be declared for a considered communication, saying that only constructs which belong to the specified type(s) are admitted for exchange. More precisely: As only representations of constructs can be exchanged via the channels of an information system, a type declaration specifies what constructs will be represented and can be "understood" by interpretation. A language, in which type declarations are made, should be called a "type definition/declaration language", but unfortunately is often called a "data definition language". This is one example of sloppy terminology which is so characteristic for the field of data processing.

Not even "type declaration language" would be sufficiently precise. As will be shown below, also other types have to be declared (on the representational level). Therefore, strictly speaking such a language is a "construct type declaration language" (CTDL). As far as only the composition of constructs by others is specified in a recursive type declaration, a graphic construct type definition language can be applied in analogy to the canonical construct representation. An example for a graphic type definition is shown in fig. 5, an occurrence of that type is represented in fig. 6, where in both figures the small box in the upper righthand corner provides a place for inserting the name of the type or the type designation (Typenbezeichnung) as we prefer to say. This "type plate" is also used in construct representation, if emphasis is put on the fact that the construct is occurrence of a particular type (cf. fig. 6 and 10).

It would be beyond the scope of this paper to discuss all the aspects involved in the concept of type in general and of construct types in particular. The one or the other will be addressed in the following paragraphs.

After this very short outline, concepts to talk about model information and a canonical representation technique are available. The concept of type has been emphasized because of its great importance for the questions of representation to be discussed in the next section.

4. Data as representations

For convenience the term "data" is used in the following instead of "digital data" indicating that only representations are considered which consist of characters (cf. [DIN]). Other representations (pictures, sounds, etc.) are not investigated with regard to their relationship to information.

Referring to the configuration of two offices with a channel between (fig. 1), let the piece of paper on which fig. 7 appears be a realization of a communication channel. The question is, whether the addressee interprets the five representations there as representations of five, four, three, two or one construct. The example suggests the answer, that the interpretation of the various representations is the subject of agreements between the communicating offices. So according to one agreement all representations might be interpreted as "number seven", according to another agreement the representation 4 + 3 might be taken for an arithmetic expression and not be interpreted as "number seven", or there might be a difference on the construct level even between a "bar seven"(7) and a "plain seven"(7), etc.

A multitude of such agreements are taken for granted in everyday communication. So in usual text the shape of the characters is irrelevant, but in mathematical texts it is not. On the contrary, you have to distinguish carefully between different fonts, because they have a different meaning which usually is agreed upon at the beginning of a paper or is default in mathematical literature. Or: In many programming languages the interspersion of blanks in some places is of no relevance, in other places it is.

These two examples may show that the relationship between information and representation (data) has to be established in advance in order to make possible mutual understanding in communication via a channel. What are the provisions to be made?

For a communication to be possible there must be a prior common background of understanding, i.e. a predefined mapping of representations onto constructs. In the course of communication further agreements may be used to extend this common background: One office passes the declarations to the other, the latter one accepts or rejects them. The declarations comprise

- construct type declaration
- representation type declaration.

Construct type declarations were discussed in the preceding section. The construct type declaration determines the constructs which can be communicated via the considered channel. The construct type declaration language is a part of the above mentioned common background.

The representation type declaration refers to a declared construct type. It determines, what are the admissible representations of constructs of this type which can be exchanged in the regarded channel. Considering the set of all representations of the occurrences of a given type we arrive at the concept of representation type (Darstellungtyp). The representation type declaration language (RTDL) is a further part of the above mentioned common background.

An example may illustrate the relationship between construct type and representation type and their respective occurrences. (The used ad-hoc languages are not to be discussed here and should be understood intuitively.) Although it is a very simple example, many figures have been necessary to depict the ideas presented sc far, which gives an indication about the magnitude of usually implied declarations.

Fig. 8 shows a declaration of the four construct types CALENDAR-DATE, MONTH-NAME, YEAR and DAY-NUMBER. The latter three are types of atoms, the first one is an aggregate type. Additionally the type composition is shown in IMC representation.

Fig. 9 shows a pertaining declaration of four representation types: MONTH REPR, YEAR REPR, and DAY REPR are the representation types for the construct types MONTH-NAME, YEAR, and DAY-NUMBER, respectively. DATE REPR is the representation type for the construct type CALENDAR-DATE.

In spite of the extensive declarations many implicit assumptions still remain: The character sets to be used, the arrangement of characters on the medium (paper e.g.) and other details. They all have to be counted to the pre-existing common background of the communicating offices.

Fig. 10 shows two occurrences of the construct type CALENDAR-DATE (and of course of the component types) and some occurrences of the representation type DATE REPR.

This example suggests that the concept of format belongs to the concept of representation type. Up to here the assumption has been maintained, that only one representation type can be declared for each construct type. This restriction should be dropped now. If multiple declaration of representation types for one construct type is provided, each of the declared representation types could be called a format (Format) in close relation to the common use of this term. Referring to the above example of fig. 9, instead of the one representation type DATE REPR we could declare three representation types (= formats) for the representation of constructs of type CALENDAR-DATE (two "positional" formats, one "key-word" format).

It can be observed that the separation of construct type declaration and representation type declaration (=format declaration) is not explicit in existing systems. The layout of the construct type declaration is often simultaneously the specification of the input and working area format. This might be a reasonable economical approach. But to understand the relationship between information and data one should be aware of the double function of such a "data definition".

Applying the view which has been presented so far of the relationship between information (constructs and construct types) and data (representation and representation types) we outline a flow of information between two offices via one channel: An office B may be requested by an office A to retrieve a construct with given properties (e.g. from a data base). Office B finds the specified construct (i.e. a representation of it), identifies the type of it, chooses one of the pertaining representation type declarations and puts a representation of the construct in question into the channel. As this representation conforms to the representation type declaration established for the regarded channel, office A is able to interpret the data (knowing the representation type and construct type).

Some reader might have noticed, that in the CALENDAR-DATE example an argumentation is missing, why the representations do not show all the details of the represented constructs (cf. fig. 10). Actually, this is not necessarily so, it only corresponds to the practice in data processing, because it is the representation which occupies storage, and not the construct. More extensive representations could be provided in a representation type declaration for various reasons (security, less extensive declarations, etc.). Of course, that would require more capacity of the involved channels (storage). In any case the question

arises, whether such a "representation" is really a representation of a construct. Strictly speaking, it is not. Only together with all specifications, which allow the interpretation of the construct, a full representation is there. Therefore a representation in the above sense shows only the individual part (Individualteil) of the represented construct, because the representational part common to all occurrences of that type is in the type declarations. This leads to the idea, that data (e.g. in "input data") usually means individual part of the full representation rather than the full representation itself. With this in mind, the use of the word "data" in the criticized term "data definition" can partly be justified: The "data definition" defines in its representation type declaration the admissible data, i.e. the admissible individual parts of construct representations. However, it should be clear by now, that the omission of the word "type" is entirely misleading.

5. Practice oriented remarks

In this concluding section some applications of the ideas about information and data as discussed above shall be tried.

First a preliminary remark: There might be the impression, that the system of IMC has been offered as a new proposal of a data model to compete with other, well known data models. That would be a misunderstanding. IMC is aiming to be a conceptual tool for speaking about information, on this level comprising the various data models. Nevertheless it is a c o n c e p t u a l model and as such offers a specific view on model information which allows to form a variety of information structures, but has its own limitations, too.

It is not the task of this paper to outline the features of hierarchic, network, relational or other data models. But it might be of interest in this context, to what these attributes refer. They refer to the so-called "data structures" which can be established in a system of the respective model and which are supported by the system's manipulation functions. With the terminology introduced above we would of course say "information structure" instead of "data structure" as meant here. Data structure in our understanding as structure of the information representation normally is left to the implementor, in order to achieve efficiency, security, or any goal else of this nature. For communication purposes the possible structures of constructs and

related questions concerning model information are of main interest: On what levels of aggregation are nominations or collections available, what are the restrictions for the nesting of constructs, are there special generic types adjusted to the application in question (e.g. "relations", which in terms of IMC are collections of equally domained nominations, called collectives (Kollektiv)), what is the support for orientation in extensive constructs, what properties can be used to address constructs (independently of their representation), and many other questions. The answers to these questions together with the pertaining operations on the constructs render a data model to be hierarchic, network or relational (or something else).

It is a matter of course, that also efficiency and other aspects influenced by representation techniques are of relevance. The problem of "redundancy" is one of them. It is not intended here to consider the benefits and the disadvantages of redundancy. But it has to be clarified, that redundancy does not refer to the level of constructs, but to the level of their representation. It has been shown, that a construct may appear at several spots as a component of an embracing construct. Spots, at which the same construct appears (necessarily or by chance) are called parallel spots (Parallelstelle). If the appearance of a construct at several spots is r e q u i r e d this has to be specified in the construct type declaration by so-called "consistency constraints" (cf. the SOURCE clause of [DDLC]). Once a consistency specification of this kind has been established, the system (as one of the communicating offices) is free to decide, whether it will store the representation of the construct each time it appears (at a parallel spot) or less often (usually once). The more often the representation is stored, the higher the degree of redundancy is said to be. It is conceivable in principle (and actually is done sometimes) that the same technique could be applied also for other than consistency-conditioned parallel spots. Such a situation is also given with the RESULT feature of [DDLC]). On the model information type level the RESULT clause specifies that the atom at the specified spot is the result of the execution of a specified procedure, which uses constructs at other spots as input. In both the SOURCE and the RESULT clause additionally is specified, whether a representation of the depending atom is maintained permanently (ACTUAL) by the system, or is made up only when required for passing it via the communication channel to the requesting office (VIRTUAL). In the strict sense, the ACTUAL feature causes redundancy. However also another, less restrictive interpretation of the ACTUAL and VIRTUAL feature is conceivable, where

the system still remains free to follow the specification verbatim (as assumed above) or to understand it only as an efficiency constraint

Doing a closer look to the discussion of redundancy (in the context of data base management systems) one encounters a system configuration which is a slight modification of that used so far. To show explicitly that one of the offices (the "system") is a computerized functional unit with a storage as a private channel (the "data base"), a diagram like fig. 11 is often preferred rather than fig. 1. With this configuration containing two channels or still better three channels (input channel, data base, output channel) we have also three places to represent constructs. If we consider a representation type declaration, the question has to be answered, what is the object channel which the declaration is applied to? As a matter of fact this is seldom clearly stated. In particular, input format declaration (e.g. sequence of atom representations) and data base format declaration (e.g. SOURCE feature, RESULT feature) are made up to one complex declaration package, the complexity of which is still more increased by packing the construct type declaration into the same package. Such declaration packages are well known under the label "schema". The consequence of such an "optimization" is a minimization of the number of characters to be written by the programmer at the expense of quality of software, in particular of clarity.

Finally some remarks on the relationship between information and data on the one hand and their manipulation on the other hand might be appropriate. It would be an obvious question to ask whether constructs or their representations are manipulated. Strictly speaking, only representations can be handled, as was stated previously. But so-called data manipulation languages do not refer to the representational level only. Primarily they are designed for the manipulation of constructs.

This will be illustrated by an example of the retrieval of a construct: The properties which are specified as parameters of a request refer to a construct rather than to a representation of it. The delivery of the found construct is done by putting it into the respective channel in an agreed representation, i.e. meeting the output format. Another example is "navigation". This term refers to moving from one spot to the other in an extensive construct. Also here no reference to the representation of this construct is involved. Only upon request the navigator gets some representation of the construct (at the spot) where he has arrived at. In case of a data base management system, he does not receive the

representation on which the retrieval has been performed, but an output representation. A counter-example, however, is a library, where the representation in the data base (room with book-shelves) is the same as in the output channel (librarian's counter).

Although a "data manipulation" language refers to the level of model information, this does not imply that no actual access to representations takes place in the system. But again, it is up to the implementor, which representations in what way he has provided to be accessed in order to execute manipulation commands. On the other hand the user has several interests to influence also the policies of representation and access. He has time, cost, and security requirements. These requirements which refer to storage and computing time exert some influence to the information level. A good choice of construct types and of manipulation functions as well as a forecast of the user's way of acting in the future (traffic density, update / retrieval ratio, etc.) should yield a balanced compromise between application adequacy and computer efficiency. However, in overall efficiency considerations the influence of storage and computing time resources will decrease. More and more it becomes evident, that we have to move from computer biased concepts, information structures and manipulation facilities to system interfaces, where more preference is given to the involved people and the intended application. This goal includes to support conceptual differentiation wherever useful. The presented view of information and data is intended to be a contribution to this goal.

References

[DIN] DIN/Fachnormenausschuss Informationsverarbeitung (FNI), DIN 44300 "Information processing; vocabulary" (German). German Institute for Standardization, March 1972

[ANSI] ANSI/X3/Sparc/DBMS Study Group, Interim Report. American National Standards Institute, February 1975

[ABN] GMD/Arbeitsgruppe fuer Betriebssystemnormung, "Terminology for the description of models of job processing computer systems" (German). GMD, St. Augustin, 1971

[PET] C. A. Petri, "Grundsaetzliches zur Beschreibung diskreter Prozesse". In: 3. Colloquium ueber Automatentheorie, Haendler, Peschl, Unger, (Hrsg.), Birkhaeuser Verlag, Basel, 1967

[DURI] R. Durchholz and G. Richter, "Concepts for data base management systems". In: Data Base Management, J. W. Klimbie and K. L. Koffeman, (eds.), North-Holland, 1974

[STEEL] T. B. Steel Jr., "Data base standardization - a status report". IFIP-TC-2 Special Working Conference "A technical in-depth evaluation of the DDL", Namur, January 1975

[ZEM] H. Zemanek, "Abstract Objects" (German). Elektronische Rechenanlagen 10/5, 1968

[DKR] R. Durchholz, W. Klutentreter, G. Richter, "Design of a data base management basic system for application programs (DAGS)" (German). In: Datenmodelle und Systementwuerfe fuer Datenbanksysteme, E. Falkenberg und W. Klutentreter, (Hrsg.), GMD, St. Augustin, 1974

[DDLC] CODASYL/Data Description Language Committee (DDLC), "June 73 Report". CODASYL DDL Journal of Development, June 1973

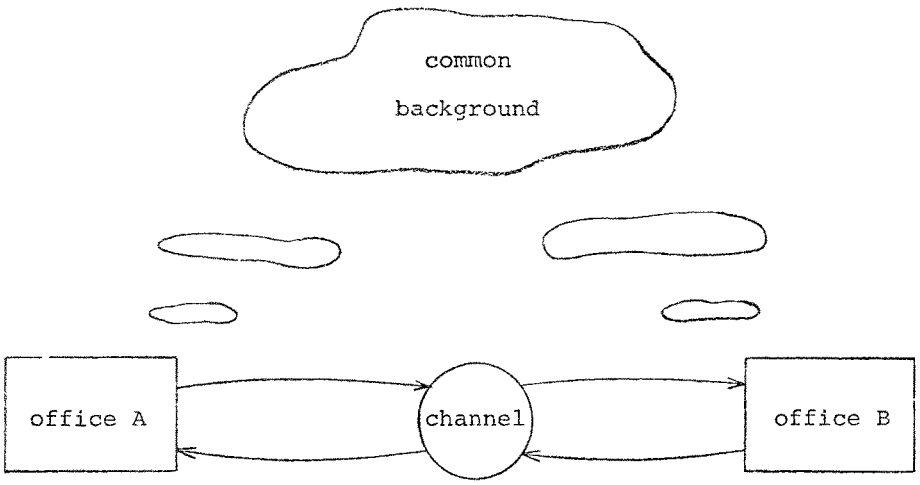


Figure 1 Configuration of communicating functional units

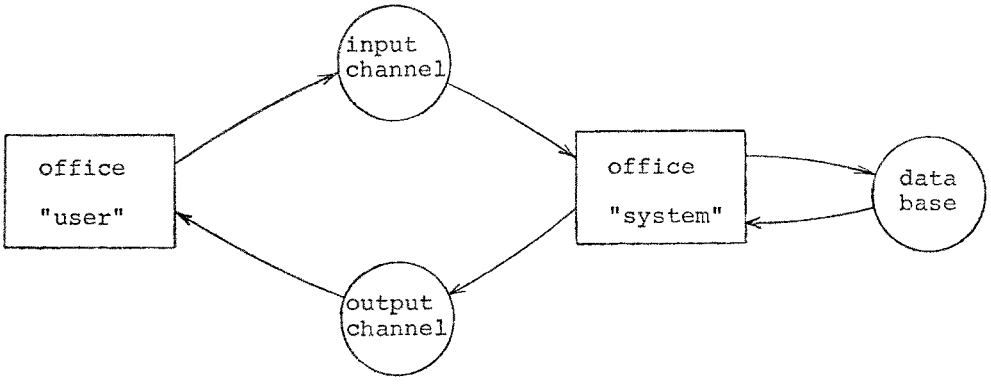


Figure 11 Extended configuration of communicating functional units

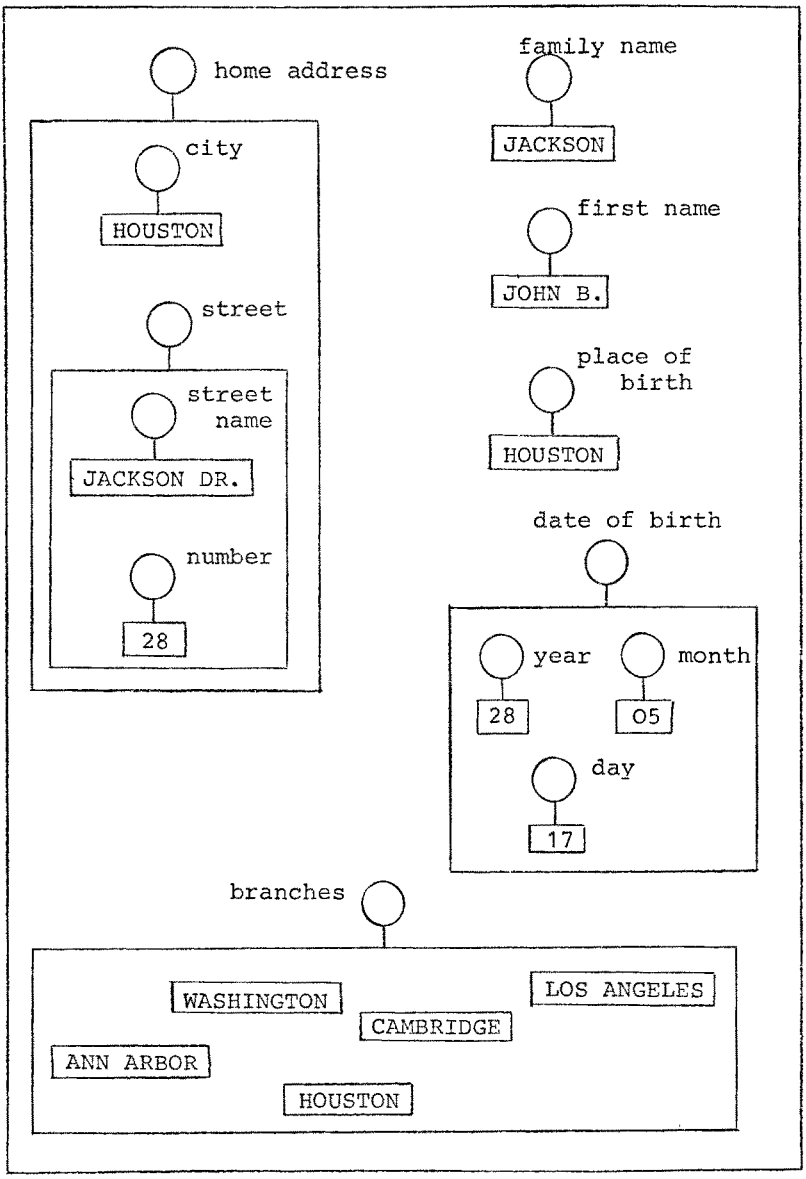


Figure 2 Constructs in IMC box representation

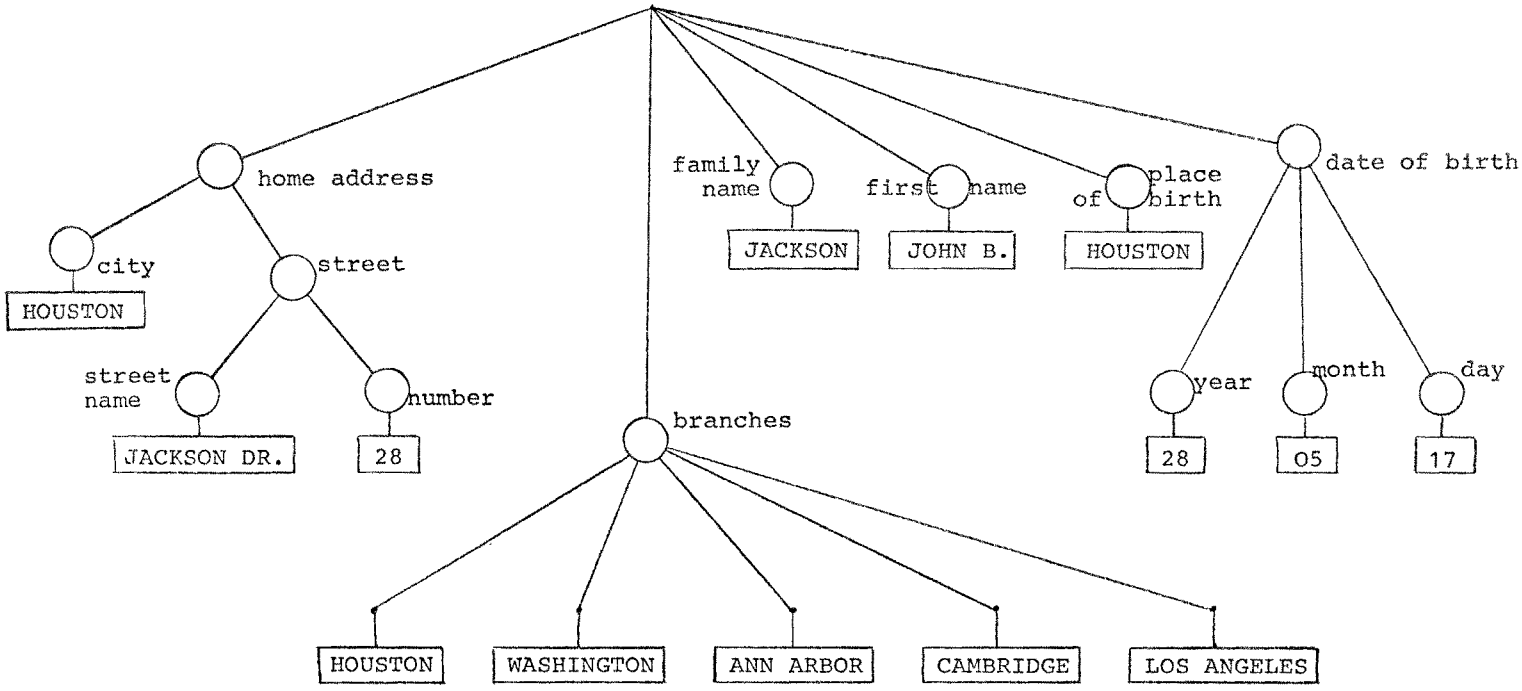


Figure 3 Constructs in IMC tree representation

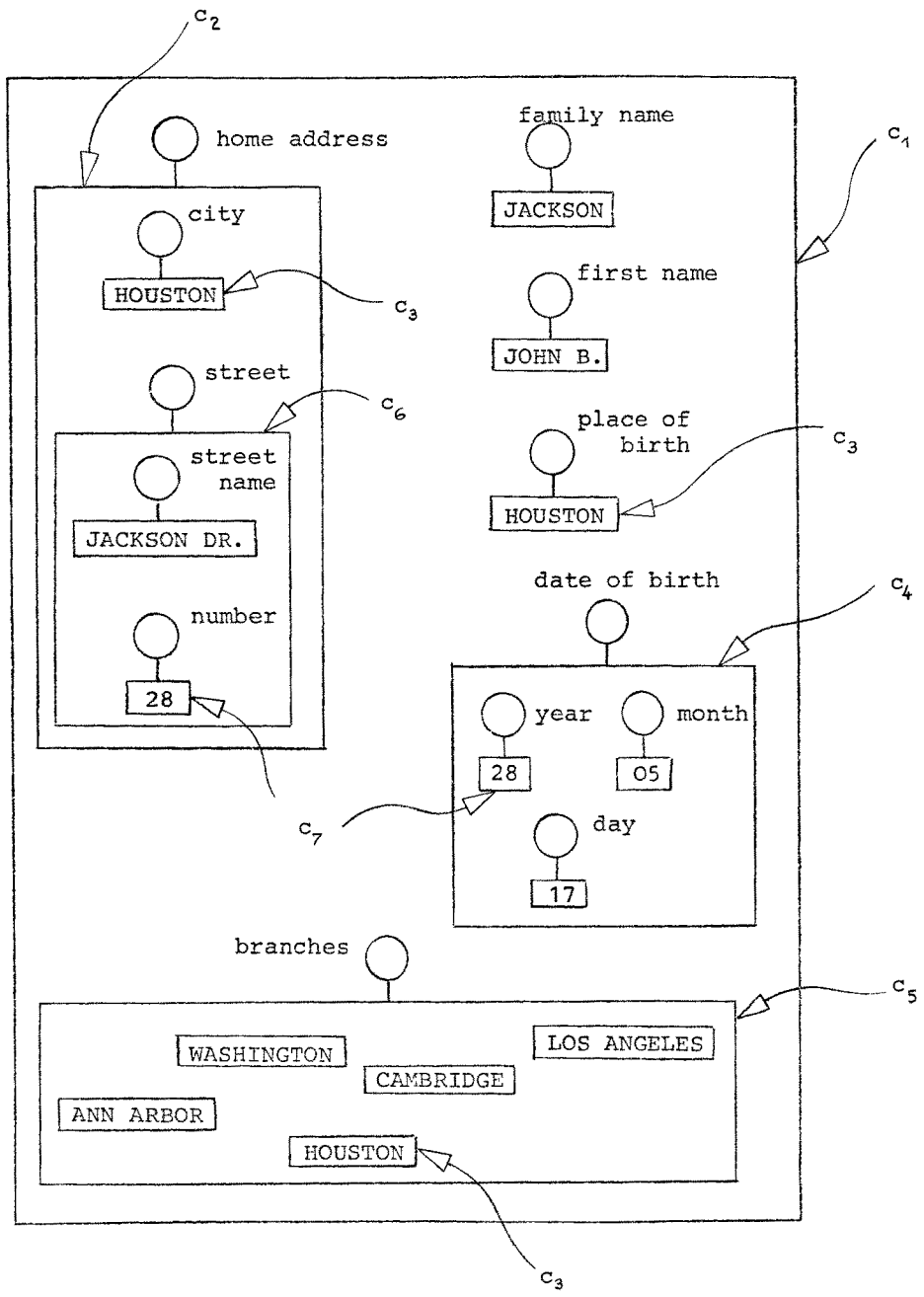


Figure 4 Construct representation of fig. 2 with additional lettering for reference purposes

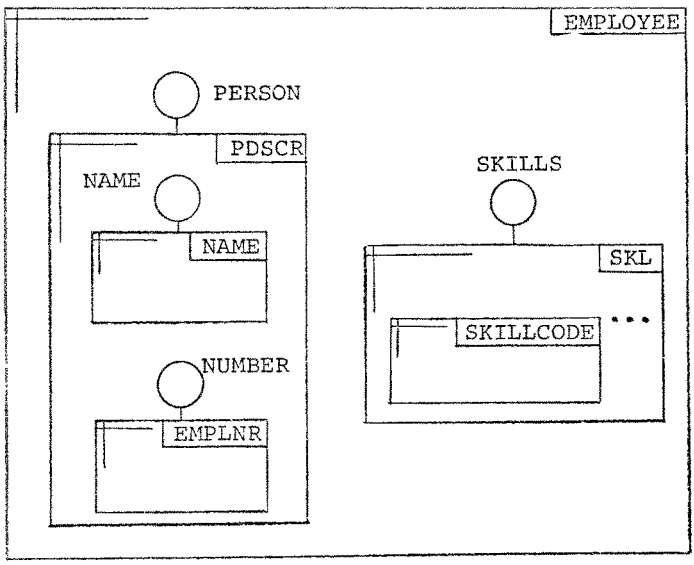


Figure 5 Graphic construct type definition

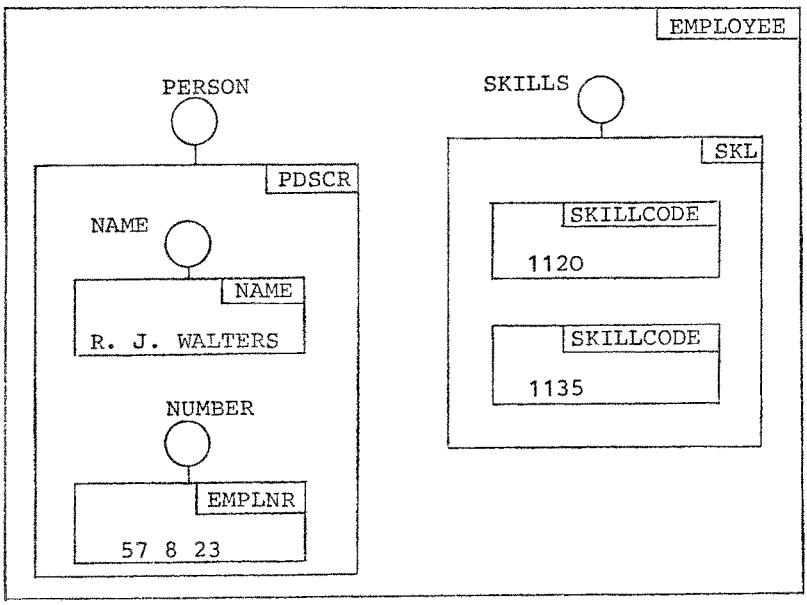


Figure 6 Occurrence of construct type defined in fig. 5

Figure 7 see next page

construct type MONTH-NAME
atom: JANUARY, FEBRUARY, ... DECEMBER

construct type YEAR
atom: 1900 ≤ INTEGER ≤ 1999

construct type DAY-NUMBER
atom: 1 ≤ INTEGER ≤ 31

construct type CALENDAR-DATE
nomination: MONTH → construct type MONTH-NAME
YEAR → construct type YEAR
DAY → construct type DAY-NUMBER
non-occurrences:

MONTH	DAY
FEBRUARY	30
FEBRUARY	31
APRIL	31

etc.

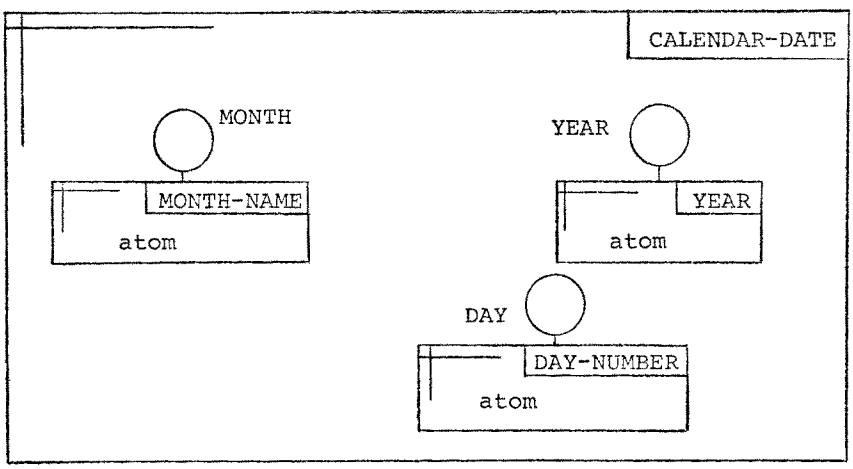


Figure 8 Construct type declarations

```
representation type MONTH REPR
represented construct type MONTH-NAME
string:  1  or  JAN → atom JANUARY
              ...
        12  or  DEC → atom DECEMBER

representation type DAY REPR
represented construct type DAY-NUMBER
string: DECIMAL representation

representation type YEAR REPR
represented construct type YEAR
string: DECIMAL representation

representation type DATE REPR
represented construct type CALENDAR-DATE
string: (DAY REPR "-" MONTH REPR "-" YEAR REPR)
      or
        (YEAR REPR "-" MONTH REPR "-" DAY REPR)
      or
        ("D:" DAY REPR /// "M:" MONTH REPR ///
         "Y:" YEAR REPR ; delimiter ",")
```

Figure 9 Representation type declarations

4 + 3

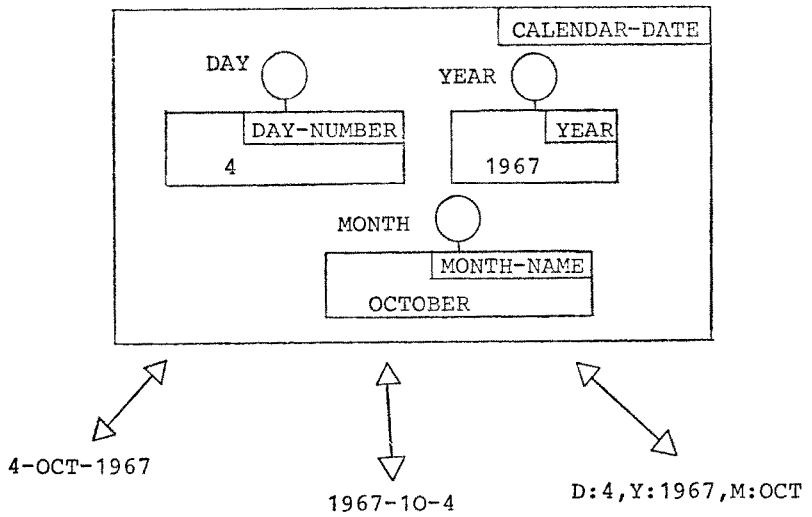
S E V E N

seven

7

7

Figure 7 Five construct representations on paper



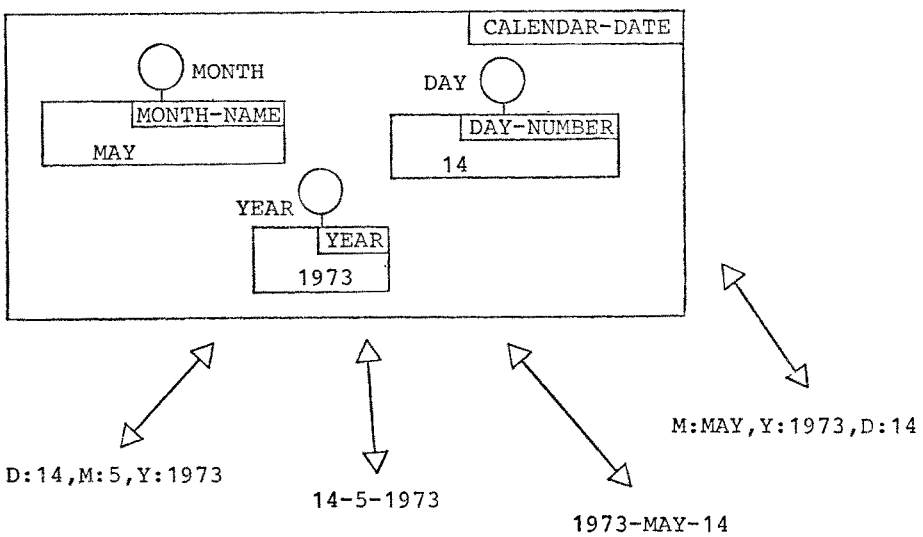


Figure 10 Construct type occurrences and representation type occurrences of fig. 8 and 9

Figure 11 see first page (fig. 1)