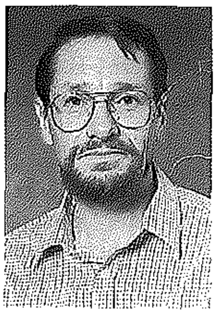
Dietrich H.Fischer
Gesellschaft fiir Mathematik und Datenverarbeitnng,
Darmstadt, Germany



Consistency Rules and Triggers for Thesauri

Fischer, D.H.: **Consistency rules and triggers for thesauri.** Int.Classif. 18(1991)No.4, p. 212-225, 11 refs.
From the point of view of an object-oriented knowledge representation language, desirable control functions for thesauri are discussed in relation to a recently published catalogue. The background of this systems-analytical examination is an experiment in prototyping a thesaurus maintenance system for an existing thesaurus using general purpose object-oriented tools. (Author)

1. Introductory Remarks Concerning Data Modelling or Knowledge Representation

1.1 Preliminary Remarks

This paper is a written version of my talk "Modelling thesauri on the basis of a frame system" given at the August 1990 ISKO Seminar on "Thesaurus Software". I was invited to give a talk on this topic as a follow-up to an article published on an experiment in object-oriented modelling of a conventional thesaurus (Rostek/Fischer, 1988). Apparently this report mainly drew attention because the prototype offered an innovative graphical interface. But the approach taken had other innovative aspects which we likewise consider interesting. In this paper, as in my talk at the seminar, I will focus on the data model or knowledge-representation aspect and especially deal with consistency rules for thesauri. In order to understand the relevance of my topic for a seminar on thesaurus software let us ask the following question:

1.2 What Does Data Modelling Have to Do with Software?

"For a long time, computer scientists have treated software development as a task primarily concerned with the construction of programs and databases. Requirements gathering and design are generally treated as preliminary steps for which there are not the linguistic tools necessary to make them "equal partners" in the software development process. We believe that the integration of AI and database research is leading to a new software development paradigm in which the software development is viewed, above all, as a task of knowledge base construction. We expect consequences of this shift to be of fundamental importance to all areas of computer science and computer engineering. The KBMS notion will play a catalytic role in effecting this shift." (Brodie/Mylopoulos (1)).

This quotation already gives shape to my answer. The process of requirements analysis should be better integrated into the software development cycle, and in order to achieve this, the gap between the system analysis language and the programming language should be narrowed or preferably eliminated. Paraphrases of this motto are

- Make your model description operational, or
- make your system specification language a programming language, or
- build your application system by
  - describing *all* needed aspects of your model and
  - compiling this description of the model into an operable system!

Such a programming language would deserve the title of a knowledge-representation language. However, the concept of a *language* seems to be too narrow, if we imagine our application system not to be a self-sufficient stand-alone system, but to be only one agent or resource in an environment shared with others. Indeed, the approach needs a kind of *knowledge-based software environment.*

At present, there are many pertinent systems which may also properly be called development systems for knowledge-directed applications. A list of names, classified as research and experimental systems, commercial application development environments, or AI languages is given in (1). Although a YES/NO-field of the list informs on which systems incorporate DBMS concepts, this can furnish no answer to the question as to when and under what circumstances these or future development systems will help to make the prototype into more than a throwaway used for the design process only.

Present knowledge-representation languages are not able to describe in a non-procedural and uniform language *all* needed aspects of a complex computer application. But the scope they cover will be enriched or extended step by step. So data models have evolved into semantic data models, which in turn are or will be superseded by a variety of what may be called knowledge-representation models or languages.

1.3 What Should a Description of an Application Model Comprise?

Traditional data models confine themselves more or less to the description of data structure and mostly leave the description of the desired or allowed processes on this structure (the semantics) to conventional application programming. Taking our object-oriented modelling tool (Rostek/Fischer (8)) and our experiments in modelling as a background, a feasible task outline would look like this:

- Description of Structure:
  - Defmition of types/classes of the objects of the domain of discourse,
  - definition of their attributes and relations (which we uniformly call *slots*),
  - definition of views (substructures).

- Description of Behaviour:
  - definition of constraints and triggers for the creation or deletion of objects and updating of their slots,
  - defmition of access paths,
  - definition of import/export converting (e.g. supporting standards like SGML)

As we see it now, the task outline has to be further extended by

- definition of presentation styles for the objects,
- definition of retrieval styles,
- definition of editing styles

in order to configure different modes of the interactive user interface.

1.4 Outline of the Class Structure of a Special Thesaurus

In any case, the structure description will be the backbone of the model to which any description of behaviour must refer. Now, with respect to thesaurus modelling what are its object classes and their structure?

The data structure and its realization is the core of any data management software and is kept secret by the software developers. With respect to the data structures of the programs INDEX, PROTERM-T and TMS hardly anything has been published (translated from C. Ritzler (5), p. 47).

One reason for this may be that the terms and concepts of the data structure of these systems are very technical and intermingled with implementation aspects regarding efficiency or ideosyncrasies of the tools used. Even if the underlying model is of the relational kind there is a gap between the cognitive model of a system analyst (or knowledge engineer or domain expert) and the list of relations or files needed (cf. (5), p. 57).

The object-oriented approach of modelling is constantly driven by the intuitive question: "What are the objects of my information interest and what are their attributes and relations?"

Of course, the main objects of the domain of discourse of a thesaurus maintenance system will be descriptors and (probably) non-descriptors. And what then are their attributes and relations? When we attempt to find names for these classes and to further specify their attributes and relations, we would have to abstract from all existing thesauri. Not only is it difficult to fmd out what is valid for all possible thesauri, but if we implement merely an abstract model this does not take into account the pecularities of existing thesauri.

Fortunately, in a so-called semantic data model we have at our disposal the modelling principle of *classification* (giving the possibility to refer to classes and their instances), as well as the principle of *generalization*. This allows us to build a hierarchy of classes along which property descriptions and behaviour can be inherited. Thus the modelling task can start bottom up by describing one thesaurus in detail, and if others have to be included, the common descriptions of corresponding classes can be factored out to common superclasses. In this way a more general model can evolve.

```
Object ()
 Synopse ()
  SfkFrame (sfkCreationInfo sfkDuplicates sfkIndexing sfkKey
                 sfkMessages sfkNames sfkStatus
                 sfkSubAspects sfkSuperAspect
                 sfkUnresolvedReferences sfkUnresolvedSlotPaths )
      SfkConceptNode (indexings subnets term)

         IZConcept (englishDenotation germanDenotation scopeNote
                   introductionNote cancellationNote cat90 cat90Inverse
                   cat91 cat91Inverse cat92 cat92Inverse thesaurusPart )
            IZDescriptor (UFhideUSEshow UFshowUSEhide
                         UFshowUSEshow descriptorNr elementOf
                         cat51 cat52 cat53 cat54 BT NT RT )
               IZDescriptorPool (elements )
            IZNonDescriptor ()
               IZComplexConcept (decompositions )
               IZSynonym (USEhideUFshow USEshowUFhide
                          USEshowUFshow )

     SfkTuple ()

        IZTuple ()
           IZRelation ()
              IZVersionRelation (date )
                 IZSupplementOrReplacementRelation
                    (cat90 cat90Inverse )
                 IZSupplementRelation (cat91 cat91Inverse )
                 IZReplacementRelation (cat92 cat92Invers )
              IZDecompositionRelation (cat51Invers cat52Inverse
                   cat53Inverse cat54Inverse decompositionOf )
           IZNote (date printFilter noteFor )
              IZIntroductionNote ()
              IZCancellationNote ()
```

Fig. 1: A Class Hierarchy for the IZ-Thesaurus

Even when describing a singlen specific thesaurus this principle of generalization can be usefully applied. I confme myself to a model of the IZ-Thesaurus (3). Fig. 1 illustrates a possible class hierarchy for this thesaurus.

Behind the bold-faced class names of Fig. 1 the slot names (attribute or relation names) of the given class are listed in parentheses. (Note: They are also known to and valid in their corresponding subclasses.) Some of them may be self-explanatory like BT or RT (denoting the descriptor-descriptor relation 'broader term' and 'related term'), while others like 'cat51' will be obscure. In any case these are just slot names, which need further specification in order to define their semantics.

Actually, the classes listed have or may have many more slots: The slots listed are only those which really store explicit object references or data. The additional slots mentioned, which we call virtual slots, are defined using the real slots; an example might be the relation RTG (related term generic) which links sibling terms with respect to the hierarchy relation. This example is resumed below.

The specialties of the class structure presented and their slots need not be discussed here; instead, let us approach the question of how the laws of their behaviour on update, i.e. their semantics, can be expressed.

1.5 Consistency Rules, Constraints and Triggers

Laws can be expressed in the form of rules or of constraints. One may say that a rule has an if-clause and a then-clause while a constraint expresses an unconditioned requirement. If in addition we take into account an authority that watches for violations and cares for law enforcement we speak of a 'trigger' representing the rule or constraint. For such a trigger it needs to be defined when the rule or constraint it represents is checked, and if it represents a constraint, what to do if the check returns false. While rules and constraints may well be considered to belong to the world of a logical formalism, triggers can be seen as their incarnation in a world of actors.

There are triggers not representing rules or constraints, but only performing some action (e.g. keeping a record) that is triggered by some event. In the object-oriented approach taken here, all rules or constraints areinterpreted or implemented as triggers. In our representation system we usc 'demon' as a synonym for 'trigger'. In addition, we use it as a generic term comprising rules and constraints. A demon is defined by an action; rules and constraints are demons which also have a condition. A rule's action gets performed if its condition is true. On the other hand, a constraint's action gets performed if its condition is false. Let us give some examples.

In order to define that e.g. the slot named BT represents a descriptor-descriptor relation we can express this by specifying that its range (or image) is the class IZDescriptor. This is a simple and standard example of expressing a constraint for the thesaurus. Implicitly, it is also a description of behaviour: On the basis of this description the system is able to protect a descriptor against getting e.g. a non-descriptor as a broader term by an update operation. So in any case, the system takes some extra action (by default or by special definition), e.g. enforcing a premature end of the update operation and giving notice of this range violation.

A pertinent example of a trigger not checking anything, but performing a side effect to preserve consistency, is the automatic update of inverse relationships. For instance, the inverse relation to 'broader term' is 'narrower term'. One may argue that this need not be expressed as a trigger. It would be sufficient to state the following rule: If the system knows that a relationship 'a BT b' holds, then it also knows that 'b NT a' holds and vice versa. Again, this is expressing a rule in the style of logic. The object-oriented representation approach seems to be more concrete with respect to knowledge organisation or even implementation: The vivid concept of a trigger here entails the idea of a constructive action (that cannot be postponed for a long time): If a relationship 'a BTb' is established (or revoked) by an update operation, then automatically the relationship 'b NT a' is also established (or revoked) and vice versa.

1.6 More about the Background and Purpose of this Paper

Now we can finish our introductory remarks which developed a tutorial setting for the rest of this paper. This will deal more specifically or technically with consistency rules for thesauri and demonstrate how we express them in our object-oriented model in such a manner that the description is compilable into a prototype. The general-purpose knowledge-representation tool used for this task is now called SFK (Smalltalk Frame Kit). It is a module added to the Smalltalk programming environment. One of its applications (coupled with a general browser module and other associated tools) is an experimental single-user thesaurus maintenance system tailored to the IZ-Thesaurus. It is operable on all workstations running Objectworks\Smalltalk (e.g. on Sparc-Workstations, Macintosh or 386-processor-based PC). We take it here as an example for the approach outlined above. Our primary intention was to explore requirements, concepts and problems of an object-oriented knowledge-representation language faced with a practical application, which had been conceived and designed as a genuine hypertext component in an author's hypertext environment already some years ago (6).

In the meantime we learned about the project ATLAS-Pflesaurus (10, 11). Its author Willenborg also developed a thesaurus maintenance system on an object-oriented basis. His approach differs from ours mainly with respect to the following points: He built a dedicated thesaurus software (using Smalltalk/V, available for the 80286-processor), without the necessity to consider the pecularities of an existing thesaurus, because the users seem to create concept nets from scratch according to the evolving needs of the ATLAS project.

The guideline of the following discussion will be a catalogue of required structuring and control functions for thesauri which was published in (9), p. 86-91. This

M.A. thesis of Dorothee Sick deals with thesauri in general and gives an in-depth analysis of the thesaurus software package INDEX (4). Although the version of INDEX with which Dorothee Sick experimented has become obsolete such that special assertions on shortcomings of INDEX will or may no longer be valid, her work was valuable to compare my analysis and description of the domain with the questions she posed to the software package.

The following discussion of her catalogue of consistency control functions is my way to express appreciation of her work, although as a result of this discussion I will show the matter to be more complex in some aspects. Furthermore, although I follow her headlines in this paper I do not back up her terminology or systematics with respect to consistency rules. Ritzler's ((5), p. 102) four point list of "all consistency rules" is contained in that of Dorothee Sick or with respect to point 4 ("Non-Descriptors must not participate in more than one relationship.") is not valid in general.

In the following my translation of the heading terms of Sick's catalogue of consistency control functions is put into quotes. In general, she explains the meaning of the terms by an example; so do I.

2. Structural Consistency

2.1 Maintenance of Inverse Relationships

It is an acknowledged standard of thesaurus software to maintain inverse relationships such as BT and NT. Of course this means that

- the inverse relationship is established whenever the corresponding relationship is established, and
- the inverse relationship is removed whenever the corresponding relationship is removed.

Dorothee Sick deals with the first consistency rule under the heading "control of incomplete references" and with the second one under "control of dangling references (Blindverweise)". With respect to binary relationships SFK can treat these rules uniformly just by declaring one relation to be the inverse of the other, e.g. by defining for the moment

```
IZDescriptor slot: #BT)
    inverseSlot: #NT; . . .
```

It has to be pointed out here that SFK (being a general tool) does not know predefined thesaurus relations, but the application-specific relations or attributes have to be defined for the application classes (IZDescriptor denotes one of them). The slot definition comprises several facets (constraint and trigger facilities); so far we only mentioned the 'inverseSlot' facet. In the course of discussing Sick's catalogue we will introduce some more facets.

Furthermore, we note that any slot (if not explicitly restricted or defined with other value collections) is set valued.

From both points it follows that SFK has no limitations with respect to structuring requirements.

Finally we note that incompleteness of n-ary relations with n > 2 will be treated separately below.

2.2 'Control of Inadmissible Relationships'

Dorothee Sick makes the distinction between direct and implicit relationships dealing separately with:

- control of inadmissible direct one-to-many relationships,
- control of implicit horizontal relationships,
- control of implicit vertical relationships,
- control of circular relationships.

The meanings of these terms will become clear in the following examples. As will be seen, there is no need for SFK to make this distinction between direct and implicit (inferrable) relationships with regard to update control.

A straightforward way to control the allowed values of a relation is to define the range of the relation. In SFK full computability is at hand to define the range. The most simple range expression is just the name of the class of the allowed values. For example,

```
(IZDescriptor slot: #RT)
    range: IZDescriptor; . . .
```

states that RT is a relation *in* the set of instances of the class IZDescriptor, i.e. with domain IZDescriptor and range IZDescriptor.

Relations *in* a set are often mathematically characterized by properties such as reflexivity/irreflexivity, symmetry/antisymmetry and transitivity. So we can state e.g. that the relation RT is irreflexive and symmetric. But symmetry is already expressed by stating that the slot RT is inverse to itself.

```
(IZDescriptor slot: #RT)
    range: IZDescriptor;
    inverseSlot: #RT;
    relationalProperties: #(irreflexive); . . .
```

The hierarchy relation is irreflexive too, but in addition it must be 'strong' intransitive and acyclic (which implies irreflexivity). Strong intransitivitity demands that no short cut must exist in the hierarchy relation, i.e. a descriptor must not simultaneously be a direct *and* an indirect narrower (broader) term of another descriptor. Acyclicity demands that a descriptor must not be its own direct or indirect narrower (broader) term. In SFK these consistency constraints can be simply stated by

```
(IZDescriptor slot: #BT)
    range: IZDescriptor;
    inverseSlot: #NT;
    relationalProperties: #(acyclic strongIntransitive); ...
```

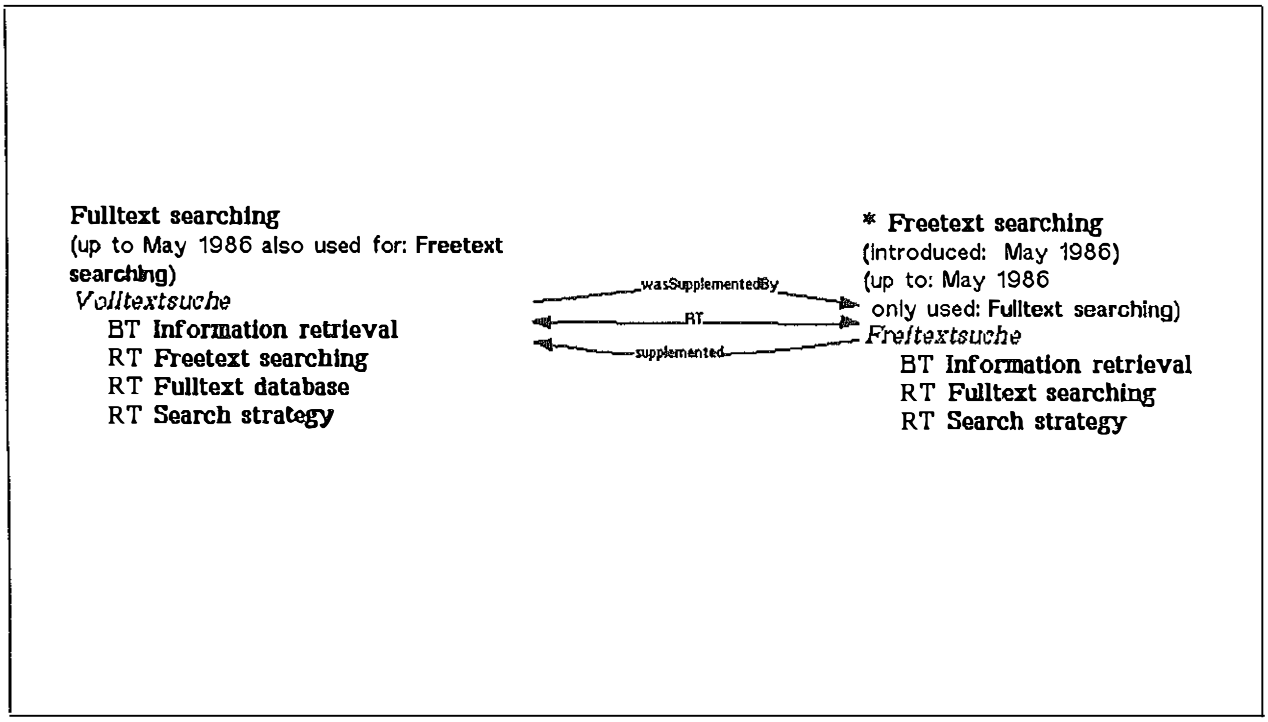


Fig. 2: An Example of Coexisting Descriptor-Descriptor Relationships

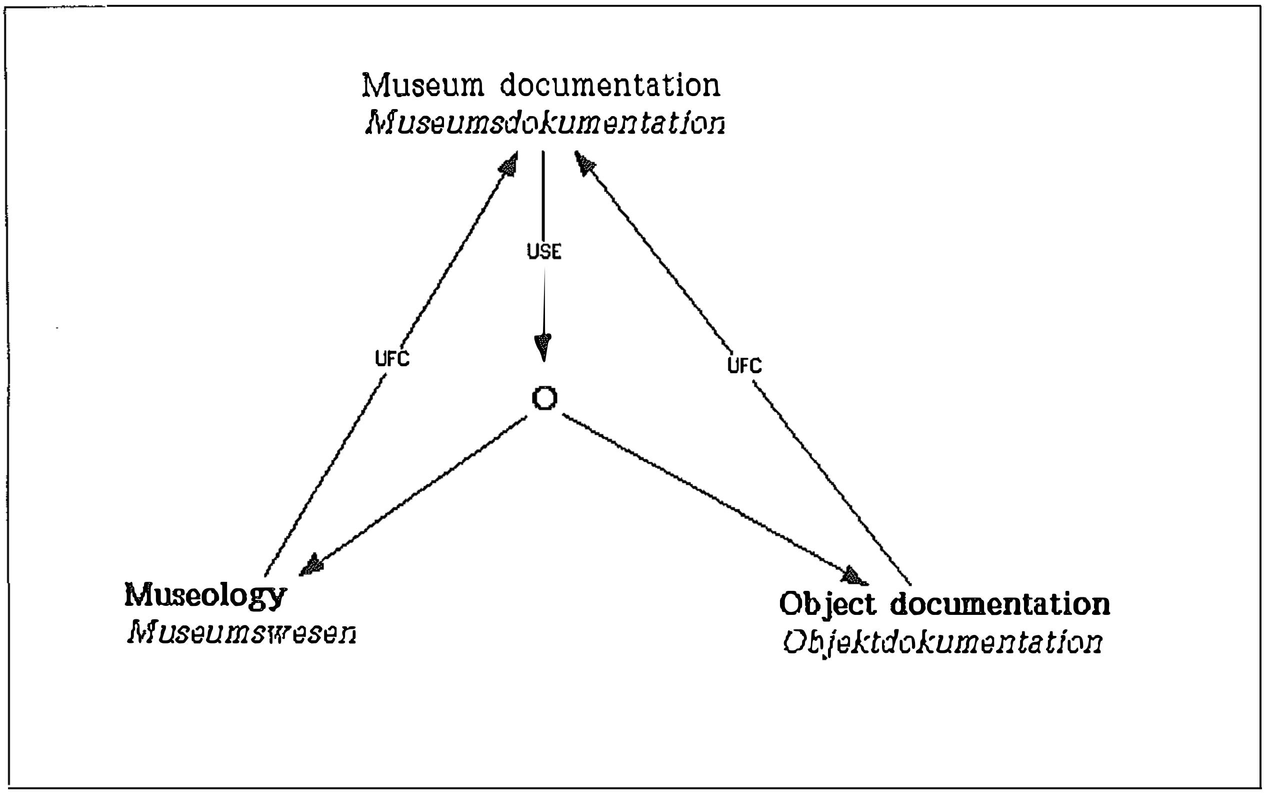


Fig. 4: A Net View of the Exemple shown in Fig. 3

So far we have already dealt with the control of "implicit vertical and of circular relationships" which Sick had missed with respect to INDEX.

It seems that under the heading "control of inadmissible direct one-to-many relationships" Sick as well as Ritzler claim that for any two given thesaurus entries maximally one thesaurus relationship between them may be valid. E.g., if two descriptors are stated to be related terms (i.e. relationship RT holds), then they must not be related by a hierarchy relationship (i.e. relationship BT or NT must not hold).

But if a thesaurus relation is just a relation between thesaurus entries, one cannot accept that thesaurus entries must not participate in more than one relationship. That law would not be true in the model of the IZ-Thesaurus (see Fig. 2). It shows an example relationship of the version relation 'was supplemented by / supplements', which coexists with an RT-relationship, saying that the descriptor "Fulltext searching" was supplemented by the descriptor "Freetext searching" in May 1986, being -- since that date -- one of its related terms.

Seemingly the model could be made simpler without loss of information by adding a time stamp attribute to the thesaurus relations storing the date of linking. There would then be no need for an extra supplement-relation. However, there is still another version relation to be represented in the IZ-Thesaurus, i.e. the replacement relation, which cannot be modelled in such a simple way.

Whether or not simultaneous coexisting relationships between two entries can be avoided by the model structure, it seems to be better to be able to control by definition which relationships shall be forbidden to coexist, i.e. which shall be incompatible. This would be a general control feature useful for any domain.

With SFK we are able to express, that e.g. a relationship RT excludes the coexistence with a BT- or NT-relationship by stating

```
(IZDescriptor slot: #RT)
    exclusiveSlotValuesAt: #(BT NT); . . .

(IZDescriptor slot: #BT)
    exclusiveSlotValuesAt: #(NT RT); . . .
```

The above expression 'exclusiveSlotValuesAt: ...' does not explicitly express what the response of the system will be when one tries to violate this law. Of course the standard reaction might be to signal an error and to roll back the update operation, but one could also imagine a different update style where the conflicting existing relationship is removed and the new one is established. In SFK this kind of behaviour can be expressed by

```
. . .
    exclusiveSlotValuesAt: #(BT NT)
        ifFalse: #update; . . .
```

Another possibility would be to refer dynamically to the actual editing style.

Under the heading "Control of Implicit Horizontal Relationships" Sick recalls the reader of the following rule: Do not establish an RT-relationship if an RTG-relationship already exists or vice versa, i.e. descriptors which are siblings with respect to the hierarchy relation shall not be connected additionally with respect to the association relation because this special kind of association can then be inferred.

In that respect we can treat direct and indirect (inferred) relationships uniformly by defining:

```
(IZDescriptor slot: #RT)
    exclusiveSlotValuesAt: #(BT NT RTG); . . .
```

In the IZ-Thesaurus the RTG-relationis not mentioned and the above constraint does not hold in general! Therefore we propose that the properties of the relations should not be built-in features of the software, but that they could be specified in a modular way.

2.3 Excursus on Virtual Relations

In the SFK-model we can defme the inferrable or virtual relation RTG on the basis of the real relations BT and NT by the following definition:

```
(IZDescriptor slot: #RTG)
    use: (SlotPath ~ #BT  ~ #NT);
    relationalProperty: #irreflexive;
    readOnly.
```

This defmition states that the values at slot RTG of a descriptor can be accessed by reading first the values at its slot BT and then collecting all the values at their slots NT which are different from the descriptor where the reading operation had started (relational property irreflexive excludes it).

So it would be simple to change IZ's policy with regard to RT and RTG: We merely need to define exclusivity as we did above and execute the following statement:

```
IZDescriptor allFrames do: [:descriptor |
    ((descriptor at: #RTG) * (descriptor at: #RT)
        do: [:value | descriptor at: #RT remove: value]]
```

which says that for all descriptors which have values in the intersection of the set of related and RTG-related values, these values must be removed from the slot RT. In this way redundant information is removed, but can be inferred on demand for display or printing.

In the same way I will show how to introduce the virtual relation 'top term' or TT of which - as Sick noticed - is not contained in INDEX:

```
(IZDescriptor slot: #allBT)
    use: (SlotPath ~ #BT);
    relationalProperties: #transitive;
    readOnly.
```

```
(IZDescriptor slot: #TT)
    use: (SlotPath ~ #allBT
        | [:f :s :value :t | (value at:#BT) isEmpty]);
    readOnly.
```

First a relation ‘allBT’ is defined to be the transitive closure of BT, then TT is defined to be the relation allBT restricted to those values which do not have a broader term.

If we further define the relation ‘is-indirect-broader-term’ by

```
(IZDescriptor slot: #indirectBT)
    use: (SlotPath ~ #BT ~ #allBT);
    readOnly.
```

we can express strong intransitivity (no-short-cut property) of the BT relation by stating the exclusiveness of the relations BT and ‘indirectBT’.

2.4 Control of Incomplete Relationships

Here we deal with n-ary thesaurus relations with n > 2, and these relations are a crucial touchstone for any thesaurus or other software package.

As examples for these relations Sick mentions the history relations ‘splitting’ or ‘union’, where e.g different concepts (e.g. countries) are splitted or united.

In the IZ-Thesaurus the following non-binary relations are used:

- The decomposition relation, which decomposes a complex concept (a non-descriptor) into (minimally) two descriptors,
  - Two version relations:
  - ‘was supplemented by’ (e.g. “up to Aug. 85 also used for . . .”)
  - ‘was replaced by’(e.g. “since Aug. 85 used . . .”)

Object documentation
Objektdokumentation
UFC Museum documentation
USE **Museology;**
Object documentation
RT **Data documentation**
RT **Documentation**
RT **Documentation of pictures**
RT **Object classification**

Museology
Museumswesen
UFC Museum documentation
USE **Museology;**
Object documentation

Museum documentation
USE **Museology;**
Object documentation

Fig. 3: Example of a Decomposition Relationship (In the printed IZ-Thesaurus the UFC-relationship for ‘Museology’ is not shown because of spatial neighbourhood to ‘Museum documentation’)

In the following I confine myself to an in-depth analysis of the decomposition relation. For an example relationship taken from the IZ-Thesaurus see Fig.3. The conceptual structure of this relationship presented in a net view is shown in Fig. 4. The German labels of the decomposition relation are BK (Benutze Kombination) at the side of the complex non-descriptor versus KB (Benutzt in Kombination)) at the descriptors side. The English notations are USE (the same as for simple non-descriptors) and UFC respectively.

Let us ask and answer some simple questions:

- *May a complex concept be decomposed into more than two descriptors?* Accidentally or not, there is no such complex non-descriptor in the IZ-Thesaurus. But let us assume that it may be allowed.
- *May a descriptor participate in more than one decomposition definition?* Actually this is true for the IZ-Thesaurus.

So the ‘degree’ of the decomposition relation is many-to-many. An SFK-slot represents a set valued function. Therefore, the decomposition relation seems to be definable by two slots in each of the participating classes.

But the matter may be more complicated: The decomposition relation would have to be a set valued relation (i.e. no function as defined in mathematics), if a complex concept may have more than one decomposition, i.e. if not only “simple” non-descriptors (represented in our model by class IZSynonym), but even complex non-descriptors may be homonyms.

Layout
(up to September 1986 used: **Layout of publications**)
USE Descriptor denoting the medium;
Design

Fig. 5: Example of a Homonymous Complex Term

A closer look at the IZ-Thesaurus shows that in fact there is one disguised candidate, cf. Fig. 5. The words “Descriptor denoting the medium” actually denote a pool of descriptors, whose members are not given explicitly but must be searched for by an intelligent lookup in the thesaurus. Therefore, the answer to the question whether homonymous complex concepts may exist is: Yes!

As a consequence, in order to represent the set valued relation USE-UFC in SFK we need a class IZDecompositionRelation, whose instances represent decomposition definitions. These decomposition definitions (represented in our model by class IZDecompositionRelation) will be intermediate objects (‘fat links’) between a complex concept and the descriptors. In the conceptual view of Fig. 4 the point of splitting of slot-

arrow USE is nothing but an instance of the class IZDecompositionRelation. If we change the presentation style of this node we get a picture as shown in Fig. 6 which will be further explained in the following.
First, let me summarize: Each decomposition definition must have

- exactly one complex concept as owner of the definition,
- minimally two descriptors into which the defined complex concept shall be decomposed,

if not, the definition has to be treated as 'object non grata'.

If I shift from natural language to SFK the definitions of all slots needed to install the decomposition relation for the participating classes IZComplexConcept, IZDecompositionRelation and IZDescriptor read as follows:

```
(IZComplexConcept slot: #USE) "the BK relation"
    range: IZDecompositionRelation;
    inverseSlot: #decompositionOf;
    minCardinality: 1; ...

(IZDecompositionRelation slot: #decompositionOf)
    range: IZComplexConcept:;
    inverseSlot: #USE;
    minCardinality: 1;
    maxCardinality:1
    componentSlot.

(IZDecompositionRelation slot: #decomposeInto)
    range: IZDescriptor;
    inverseSlot: #UFCdefinitions;
    minCardinality: 2;
    componentSlot.

(IZDescriptor slot: #UFCdefinitions)
    range: IZDecompositionRelation;
    inverseSlot: #decomposeInto.

(IZDescriptor slot: #UFC) "the KB relation"
    use:
    (SlotPath ~ #UFCdefinitions ~#decompositionOf);
    readOnly.
```

Minimal cardinality constraints -- as expressed above -- are treated by SFK (if not otherwise declared) as *soft* constraints, i.e. they are only checked on removal of a value, and on violation result in a warning, but do not induce a rollback. The SFK-method componentSlot tightens the soft constraint and makes it a hard constraint such that SFK installs for any decomposition relation instance a deletion dependence from its constituting concepts (its components). In other words, the decomposition relation object is automatically deleted, i.e. all references to it are deleted, whenever a remove operation violates the minimal cardinality constraint or whenever any terminating 'main transaction', in which such ȧn instance was created, detects that the related minimal cardinality constraint is not fulfilled. This guarantees that there is no 'incomplete relationship' as understood by Sick!

On the other hand, (if we do not add further behaviour descriptions) we do allow that there may exist complex concepts which do not have a decomposition definition. However, if we also interpret a download of an alleged complete thesaurus or an editor's terminal session as a 'transaction', then our patience with respect to the existence of complex concepts violating their minimal cardinality constraint may be wearing thin at the end of that transaction.

Consequences are: First, there may be constraints that need a short term reaction lest we are drowned in inconsistencies, and second, the system must be able to tolerate incompleteness of information or even inconsistencies until the complex task is finished or alternatively give us cooperative reminders when appropriate.

I used the term transaction, an important concept from database management systems. Indeed, such a mechanism is needed for thesaurus software. SFK supports only one transaction feature, i.e. the 'atomicity': A procedure is said to be 'atomic' if it is done completely satisfying all *hard* constraints or alternatively does not leave any footprints.

SFK can support this kind of atomicity property of complex operations on the object net. Even filling a slot internally is a complex procedure which consists of several suboperations which are programmed descriptively by the application designer when he defines the slot facets at the class level. E.g. adding a value to a slot (instance level) is an atomic procedure in SFK. Furthermore, creating frames and updating some of their slots can be encapsulated in SFK and thus installed as an atomic operation. One way to do this is to define a slot (real or virtual) which accepts all input of the complex operation. Filling such a slot may be compared to telling the whole story to the porter or the receptionist who then seemingly makes all arrangements for you although he passes the information to the pertinent experts who really do the job; finally he gives a response to you: Committed, not committed, or committed with some proviso (e.g. unfulfilled soft constraints).

In order to illustrate this, we resume our small-sized example: Let us assume that the user is just editing the descriptor 'Museology' and that the definition given by the string 'Museum documentation USE Museology; Object documentation' needs to be added to the thesaurus. This is exactly the situation on batch input from the IZ-Text file (there may even be more information to add with respect to the decomposition relation or complex concept, cf. Fig. 7, at category 51). If you look also at Fig. 3 you will see that the definition to add just seems to be the value at slot UFC! But according to the above definition the slot UFC is a read-only slot not accepting any input.

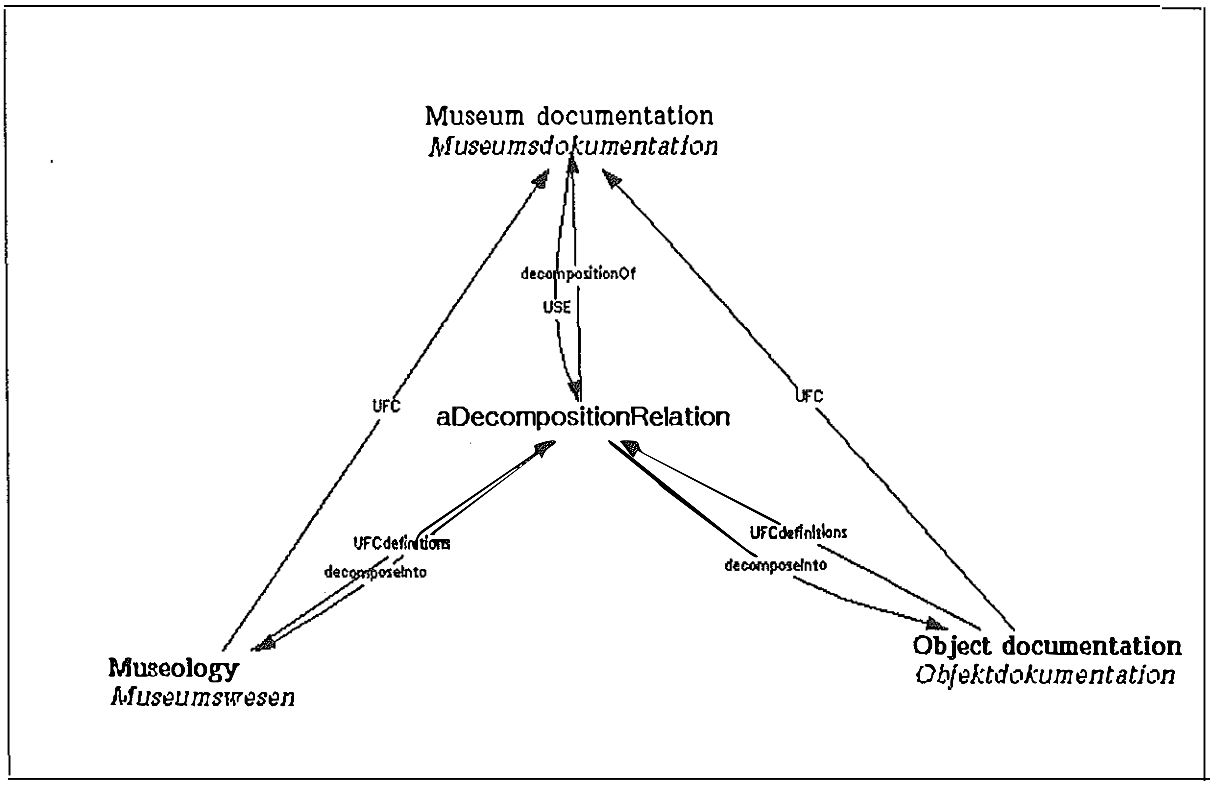


Fig. 6: A more detailed Net View of the Decomposition Relationship shown in Fig. 4

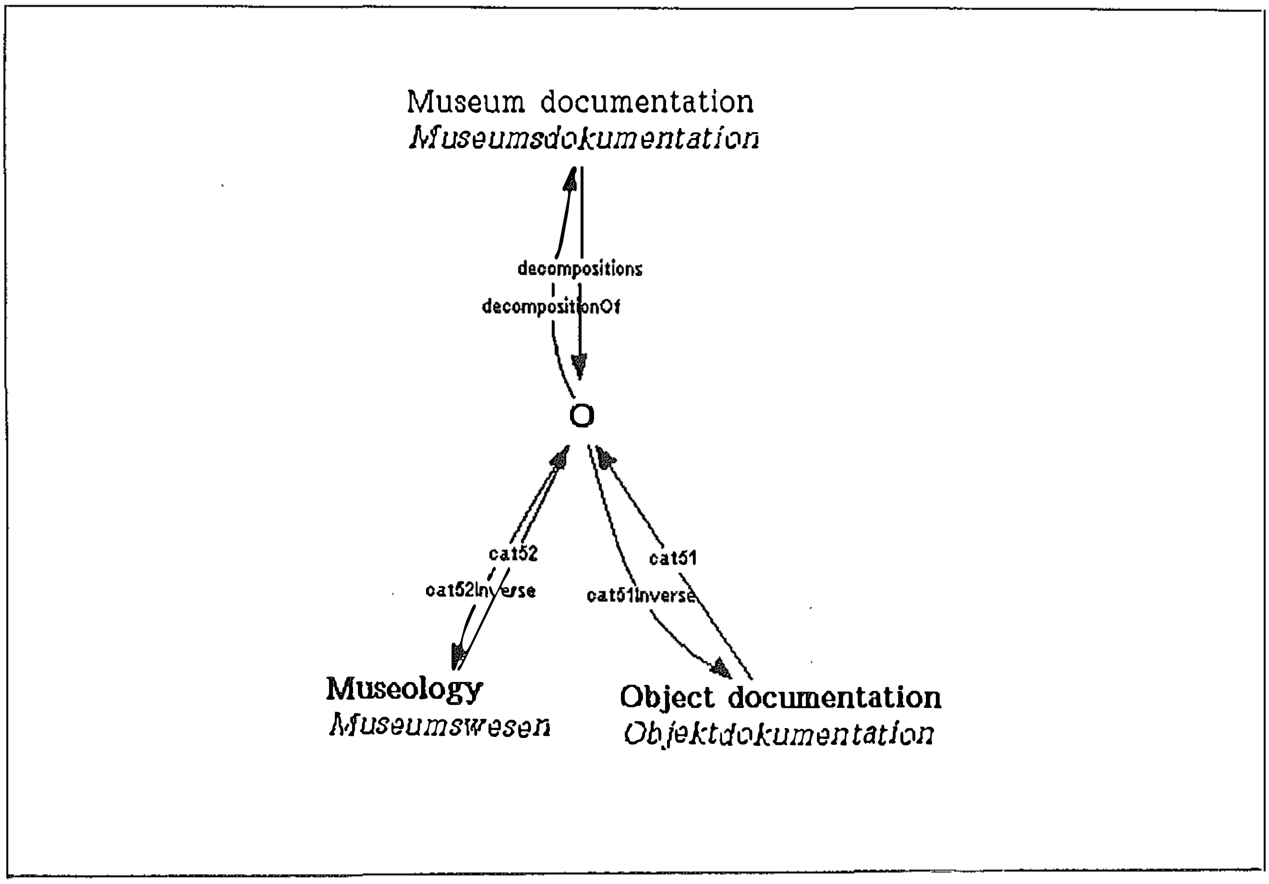


Fig. 8: The Real Storage Structure of the Decomposition Relation Example

```
+++
10*D0312++
20*Information Retrieval++
21*Infonnation retrieval++
30*Recherchieren++
40*Freitextsuche++
40*Rechercheergebnis++
40*Recherchestrategie++
40*Volltextsuche++
50*Infonnationswiedergev/innung++
50*Retrieval++
51*Retrievalsprache=Information Retrieval; Kommandosprache
    ?bis ++*Febr. 86 benutzt: %Kommandosprache%?++
60*Infonnation Retrieval System++
60*Infonnationsvermittlung++
60*Recherche++
60*SDI++
70*Vorgang der maschinenunterst tzten Suche in Datenbanken++
90*bis Mai 86 benutzt: %Maschinelle Recherche%++
+++
```

Fig. 7: Example of IZ-Text-File Record Format

Now let us change the definition of slot UFC:

```
(IZDescriptor slot: #UFC)
  use:
  (SlotPath ~ #UFCdefinitions ~ #decompositionOf);
  virtualSlotInputCompilingFromStringByRule:
      #UFCdecompositionDefinition
```

Thereby we install a "porter" or "receptionist" at the slot: The SFK-method virtualSlotInputCompilingFromStringByRule: defines two 'demons' for the slot: One is called on adding a string at the slot and the other on removing a string from the slot. Let us explain the demon called on adding (which in fact is an 'insteadAdd-demon'): Instead of adding this string to the slot's value collection, it parses and transforms the input string into frame objects according to a grammar tool attached to the class. The parsing process has access to the object net, so some actions of the grammar, referenced by # UFCdecompositionDefmition, find or create objects from accepted parts of the strings, e.g. retrieve or on absence generate the complex concept 'Museum documentation' and the descriptor 'Object documentation'; furthermore, an instance of IZDecompositionRelation is generated and the two descriptors are added at its pertinent slot 'decomposeInto'; finally this defmition object is added at the slot USE of the complex concept 'Museum documentation'. If the parsing process fails or some of the mentioned slot addings (subtransactions) of the 'insteadAdd-demon' fail, this is signalled to the main transaction which then rolls back.

This style of referencing an ATN-like grammar is at least a compact kind of procedural application programming. One benefit of this style is its modularization of the description of the model into classes, slots, slot facets, grammars, rules and transforming/mapping actions. *Downloading* the whole thesaurus from a text file as shown in Fig. 7 can be written as a (long) transaction in a very compact and modular way as a sequence of subtransactions, which may also be used in interactive update operations.

In our example an explicit slot filling at the instances of the 'link'-class IZDecompositionRelation can and should be made a reserved task for the system. The user communicates with the system by mouse selections or string input for whole transactions.

The asymmetry with respect to slots USE and UFC introduced above can to a certain extent be eliminated if we allow for the string input format e.g. 'Museology; Object documentation' and add the following definition at slot USE of class IZComplexConcept:

```
inputCompilingFromStringByRule:
    #USEdecompositionDefinition
```

Then it is even possible to add a complete definition at the real slot USE. The SFK-method

```
inputCompilingFromStringByRule:
```

defines an input conversion routine which in spite of the range definition of the respective slot preliminarily receives strings as slot input. In contrast to the demons for slot UFC the referenced parsing and compiling process does not play the role of an 'instead-demon', but is a special kind of a 'beforeAdd-demon' that generates and delivers the relation instance (which references the two descriptors) to be added at slot USE.

Implicitly, we now have explained the tools and methods used to map input text to structured objects. There is an ergonomic need not to free the editing user from rigid field and subfield filling, and then let him edit the thesaurus on views as presented in the figures showing the official text view (Fig. 3, 9, 10) according to a WYSIWYG-style (What You See Is What You Get).

Let us resume listing the constraints for the decomposition relation: We allowed complex concepts to have more than one decomposition definition; but of course if there is more than one definition the definitions should be different! But: When are decomposition definitions different or equal? Let us look at the following examples:

- Museum documentation
  USE Museology; Object Documentation
- Museum documentation
  USE Museology; Documentation

or

- Searchtree
  USE Interactive videotext; Search strategy
- Searchtree
  USE Interactive videotext; Search strategy;
  Classification system

Considering these examples we realize that we need to be able

- to define equality or compatibility of complex objects in a flexible way,
- to define what shall be done when two objects are detected to be equal or compatible in an update operation.

As a solution to this problem we add the following two value equality specifications to the description of slot USE of class IZComplexConcept:

```
    valueEqualityCheckOn: #(decomposeInto)
        ifTrue: #rollBack;
```

This definition refers to the slot named #decomposeInto defined above for class IZDecompositionRelation. It says that if a candidate definition has the same decomposition into descriptors as one of the already known definitions then the candidate value is deleted.

```
valueEqualityCheckBy: [:f :s :candValue :t | | oldValue |
    oldValue := t oldValue.
    ((oldValue at: #decomposeInto)
        includesAll: (candValue at: #decomposeInto))
        or:[(candValue at: #decomposeInto)
        includesAll: (oldValue at: #decomposeInto)]]
ifTrue: #warning.
```

This equality-check demon examines whether the candidate definition has an overlapping decomposition with any of the known (old) definitions. If the answer is yes, the demon does not cause a rollback of the adding operation, but adds a warning note to the complex concept that it has overlapping decomposition definitions. Then this complex concept can later be edited e.g. as a result to a query asking for anywarning note objects.

We now conclude our investigation of the decomposition relation. Actually, the IZ's use of it is even more complex because for each participant (descriptor or complex non-descriptor) in such a definition there is a binary valued attribute encoding whether this participant will show the definition when it is printed in the official thesaurus. In addition, the downloading process from the original text format needs a kind of unification process on these values because the full information comes from different places of the text file and even contradicting encodings are possible and have to be checked. All these subtleties have been represented with relative ease in the SFK-model.

For those who still want to learn more details: In our model the show/hide-information for each participant of the definition is represented by different slots according to category numbers used by IZ (cf. example record format of Fig. 7). Also look back to Fig. 1. The slot USE of IZComplexConcept defined above is an alias for the real slot 'decompositions' while the slot 'UFCdefinitions' of IZDescriptor is a union of the real slots 'cat51', 'cat52', 'cat53', and 'cat54'; their respective real inverse slots at IZDecompositionRelation are 'cat51Inverse' to 'cat54Inverse' which are virtually united by the slot 'decomposeInto'. Figures 4 and 6 (which are hardcopyies from the screen) show a view of something which has the real structure as presented in Fig.8.

3. Domain Specific Constraints

What Dorothee Sick calls "logical consistency" we call "domain specific constraints". Her examples for this point are apparently related to special check procedures of the INDEX software. All INDEX thesaurus entries may have predefined fields called 'group', 'facette', 'class of word' and 'language'. The admissibility of a thesaurus relationship between two entries can be controlled depending on identical or different values of these fields belonging to the two candidate neighbours. E.g. a translation relationship is permitted only if the two entries have *different* values in the 'language' field. And a synonymy or quasisynonymy relationship is permitted only if the corresponding values of all four fields are *identical*, whatever they are.

We just give an example for a simple constraint which does not seem to be checkable by such a special control mechanism: If you think of the thesaurus entry attributes 'date of introduction' and 'date of cancellation' then for all entries the date of introduction has to be earlier than the date of cancellation. SFK expresses such a constraint simply by specifying 'range conditions' as follows:

```
    (IZConcept slot: #dateOfIntroduction)
    range:
    Date & [:theEntry :givenValue | | d |
                (d := theEntry dateOfCancellation) isUnknown
                        or: [d > givenValue]]

(IZConcept slot: #dateOfCancellation)
    range:
    Date & [:theEntry :givenValue | | d |
        (d := theEntry dateOfIntroduction) isUnknown
                        or: [d < givenValue]]
```

The range definition here specifies that the values of these slots (attributes) must be dates (instances of a class Date) and fulfill the test coded between the brackets. It simply states that the date of cancellation (introduction) is unknown or greater (lesser). The demand to be an instance of class Date is itself the constraint that the presented value is a well formed date, which can understand a test procedure '<' or '>' according to date semantics. Such a date is not a string, which would respond to a test like '<' in quite a different way than a date (lexicographic order versus time order). In order to be able to present a string denoting a date as a value to these slots we can attach to them a converting procedure. Furthermore, we would like to point out that the slots must not have more than one value. So a more refined specification of slot 'dateOfIntroduction' reads like this:

```
(IZConcept slot: #dateOfIntroduction)
    minCardinality: 1; maxCardinality: 1;
    range:
 Date & [:theEntry: givenValue | | d |
        (d := theEntry dateOfCancellation) isUnknown
                        or: [d > givenValue]]
    inputConvertingFrom: String by: [:givenString |
        Date readFrom: givenString]
```

Of course in addition one could demand that each descriptor should have an introduction date, then the formulation of the cancellation date would be simpler. Or one could define a kind of default value if it is not stored explicitly. So we would get

```
(IZConcept slot: #dateOfIntroduction)
   minCardinality: 1; maxCardinality: 1;
   range: Date & [:theEntry :givenValue |
      theEntry dateOfCancellation > givenValue]
   inputConvertingFrom: String  by: [:givenString |
      Date readFrom: givenString];
   ifNeeded: [IZConcept defaultDate].
```

The cancellation dates even introduce other constraints: E.g. for a cancelled descriptor establishing additional thesaurus relations must be prohibited, i.e. cancelling means to 'freeze' the descriptor (not to throw it away). Such a behaviour may better be handled by a kind of version mechanism, although this can also be simulated by simple SFK-procedures attached to the slots (so called 'demons'). Another solution would be to model mutation, and for that purpose introduce new classes for cancelled terms. This enables providing them with a behaviour of their own.

4. Typos and Duplicates

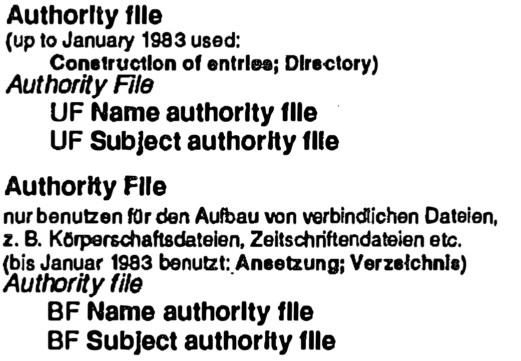
Dorothee Sicks's heading term 'input control' does not explicitly say what is controlled, but at best when control is exhibited. In fact, Sick treats the problem of avoiding duplicate entries of terms (descriptors or non-descriptors) and the problem of creating entries with misspelt term names. In other words, we address the identity problem of a representation system: The ultimate aim is to get a one-to-one-correspondence between world objects and their representations in the system. The problem arises when the input conversion action results in more than one representation in the system for one external object, or only one representation for different external objects, or a representation not referencing any external object.

The last case is equal to the problem of **misspelled term names** that are not hits for other existing term names. It could be tackled with some expense using a spelling checking device. There is another possibility to handle the problem, if there is redundancy to be exploited. For example, when the IZThesaurus is downloadedfrom the text file (cf. Fig. 7), which is grouped into descriptor records, any descriptor frame created gets the status 'filedIn' only when and if its description has been processed. However, it does not get this status if it has been created when the system processes a reference to the descriptor within another descriptor's record. If any reference to descriptors was a typing error, then after completing the download there exist descriptors not having the status 'fileIn', and they can easily be found by a query. Such a control was not feasible for non-descriptors because there were no separate redundant descriptions of them in the original IZ text file.

A further device exploiting redundancy can be installed in SFK by counting the number of times a relationship between terms is stated: When a user tells the system that descriptor A is narrower than B and later, in a redundant way (as in the download process mentioned above) the system is told that descriptor B is broader than A, then the system recognizes redundance and ends the operation prematurely. Depending on the user-defined editing style, it can also display a message: "Relationship ... already known!", as the inverse relation has already been installed on the first event. Instead of or in addition to emitting the message a counter for this relationship could be increased by one. In order to install such a device we would have to model all relations using link classes, i.e. representing each relationship by a frame which is deletion-dependent from the nodes (the concepts) it connects. An example for such a link class was given when I treated the decomposition relation.

A simplistic way of controlling **duplicates** is to prevent the user from creating a duplicate. For example, the user gets a blank input form where first of all the key property 'denotation' has to be filled in. Before the user can key in any further potentially conflicting data, the system searches for already existing data belonging to the key, and if found, fills it into the input form.

But things look different, when e.g. in a *batchprocess* a duplicate is filed in which was created independently such that data of the duplicates have to be merged. For such cases SFK allows to define merge rules.

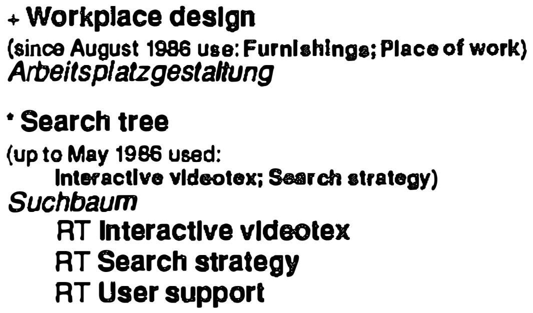


Authority file
(up to January 1983 used:
Construction of entries; Directory)
Authority File
UF Name authority file
UF Subject authority file

Authority File
nur benutzen für den Aufbau von verbindlichen Dateien,
z. B. Körperschaftsdateien, Zeitschriftendateien etc.
(bis Januar 1983 benutzt: Ansetzung; Verzeichnis)
Authority file
BF Name authority file
BF Subject authority file

Fig. 9: Example of English/German Twins

And what about the scope of the key control? It has to cover descriptors *and* non-descriptors. Things get even more complex with a multilingual thesaurus if we are not satisfied with crutches like uppercase for German and lowercase for English terms or addings for homograph resolution (cf. (9) , p. 78 and 108). Confer Fig. 9, which is taken from the IZ-Thesaurus and where the first entry is the English entry and the second is its German twin. This shows that the key must have a clear scope which may pertain to several classes. It would help to have 'language' as a second key attribute.



+ **Workplace design**
(since August 1986 use: Furnishings; Place of work)
Arbeitsplatzgestaltung

• **Search tree**
(up to May 1986 used:
Interactive videotex; Search strategy)
Suchbaum
RT **Interactive videotex**
RT **Search strategy**
RT **User support**

Fig. 10: Example of Term Mutations

A wholly different feature could complicate our simplistic view of the world: Let us look at Fig. 10 which again shows two entries from the IZ-Thesaurus: The cross in front of the bold-faced entry name "Workplace design" indicates that this is a cancelled descriptor, and the following text informs us that in August 1986 this descriptor was replaced by the combination of "Furnishing" and "Place of work". In fact, by that declaration the cancelled descriptor was replaced by a so-called complex concept with the same denotation. In other words, the descriptor "Workplace design" was changed into a non-descriptor, specifically a complex non-descriptor, which refers to minimally two descriptors by the decomposition relation. The second example "Search tree" on the other hand was changed in May 1986 from a complex non-descriptor into a descriptor. (It is reasonable to conserve this mutation information if it cannot be guaranteed that the documents indexed using this thesaurus have been reindexed after the thesaurus was updated.)

The question here is: Shall the cancelled entry with all its data still be accessible as an older version? Can thesaurus administrators make profitable use of on-line search in older versions of the thesaurus which have the same conceptual net space as the actual version, i.e. are not held in different files? In such a space the coexistence e.g. of a cancelled descriptor and a living non-descriptor with equal denotation would have to be tolerated. Again, this would be a good reason to have separate classes for cancelled terms and to modify the schema shown in Fig. 1!

There is a third argument for being more tolerant with respect to duplicates: Think of cooperative thesaurus or lexicon writing: It may be useful before "imprimatur" that for comparison and discussion purposes incompatible and rivalling entries should be allowed.

From this discussion it follows that duplicate control in the sense that all duplicates are simply rejected apparently is a demand or device too rigid for an advanced thesaurus or terminology system.

The experiences gained from modelling the IZ-Thesaurus have not only influenced the further development of SFK, but are also applied to a comprehensive model of a terminology lexicon (2). Key control for complex keys was implemented in such a way that duplicates, triplettes, etc. may be allowed, but are well controlled. In the lexicon model the system maintains a homograph number in addition to an intellectually controlled homograph resolution term. It is possible to download several subnets (e.g. thesauri from different sources) into one common model with or without merging the information according to the actual downloading style.

SFK does not yet have the language to specify these editing or retrieval styles in a general and ergonomic way. Furthermore, the object presentation in different styles still needs conventional programming. SFK does not offer concurrent updating and still waits for an interface to a database system in order to manage more than a few thousand terms. The browsers, i.e. the user interface tools for browsing and updatingthe instances are still in the first prototype stage.

5. Conclusion

We have argued in this paper that even dedicated thesaurus software needs flexibility and extensibility to deal with special constraints or rules of the domain which emerge when the model approaches a richer real world or when users ask for more 'intelligence' exhibited by the system. To be more general, software construction should be based on compilable, readable and maintainable descriptions of the structure and update behaviour of the domain of discourse. We explained some already operable steps in this direction.

Acknowledgement
I thank my colleagues Wiebke Möhr and Lothar Rostek for valuable comments.

Notes

[1] Those who will compare it with the hierarchy presented in Rostek/Fischer (7) will not only find different names (English instead of German), but also differences in structure. Let us take it just as an example of a 'schema' modification showing that the process of structure design is an evolutionary process. I will indicate at some points in this paper that a more elaborate, presumably preferable representation can be given. This concerns versions and multilinguality. The IZ-Thesaurus is in principle monolingual; a translation exists which except for two terms represents a one-to-one relation. In the meantime we have designed a model for a multilingual lexicon that has many more classes (2).

[2] In standard data modelling terms the denotation of the entries has to be declared to be a *key* attribute or key slot in order to control the uniqueness of its values.

References

(1) Brodie, M. L., Mylopulos, J.: Knowledge Bases and Databases: Semantic vs. Computational Theories of Information. In: Ariav, G., Clifford, J. (Eds.): New Directions for Database Systems. Norwood (NJ): Ablex 1986. p.186-218

(2) Fischer, D., Möhr, W.: Lexikon-Redaktion: eine Herausforderung fuer Computer-Assistenz beim Publizieren. In: Der GMD-Spiegel1 '91, St.Augustin: Ges.f.Math. u. Datenverarb. mbH (GMD) März 1991, p. 40-45

(3) Ges. f. Inform. u. Dokum. mbH: Deskriptorenliste zum Bereich Informationswissenschaft und -praxis: Stand: Juli 1987, Bearbeitung: Inform.zentrum f. Inform.wiss. u. -praxis (IZ) Frankfurt: GID, 1987. p.109

(4) Lukas, E.: INDEX - ein Programm zur Erstellung von Wörterbüchern und Dokumentationssprachen auf Personal-Computern. In: Nachr. Dok. 39(1988)p.253-256

(5) Ritzler, C.: Vergleichende Untersuchung von PC-Thesaurusprogrammen. Diplomarbeit, Fachhochschule Darmstadt, Fachbereich Inform. u. Dok. (1989)132 p.
(see also Int.Classif.17(1990)No.3/4, p.138-147)

(6) Rostek, L., Fischer, D.: An Author's workstation: visions, views and activities. In: Miller, J. (Ed.): Protext II: Proc. 2nd Int. Conf. on Text Processing, 23-25 Oct., Dublin. Dun Loaghaire; Dublin: Boole Press 1985, p. 82-95

(7) Rostek, L., Fischer, D.: Objektorientierte Modellierung eines Thesaurus auf der Basis eines Frame-Systems mit graphischer Benutzerschnittstelle. In: Nachr,Dok. 39(1988)p. 217-226

(8) Rostek, L., Fischer, D.: SFK: A Smalltalk Frame Kit - Concepts and Use. To appear.

(9) Sick, D.: Aufbau und Pflege komplexer natürlichsprachig basierter Dokumentationssprachen (Thesauri): Aktuelle Tendenzen und kritische Analyse einer ausgewählten autonomen Thesaurus-Software für Personal-Computer (PC). MA-Abschlußarbeit, Fachrichtung Informa.wiss. Univ.des Saarlandes, Saarbrücken (1989), 136 p.

(10) Willenborg, J.: Pflesaurus - ein System zur Erstellung und Weiterentwicklung von Thesauri. Diplomarbeit, Januar 1991, Techn. Universität Berlin, Fachbereich Informatik, Gruppe Wissensbasierte Systeme (WSB), Franklinstr. 28/29, 1000 Berlin 10, 97 p.

After having finished this paper I received:

(11) Mengel, A.: Thesaurusrelationen, Konsistenz, Inferenz und Interdependenz. Forschungsbericht Nr. 91-5, Techn. Universität Berlin, Interdisziplinäres Forschungsprojekt ATLAS, Hardenbergstr. 28, D-1000 Berlin 12. 22 p.

Author's address:
Dietrich H. Fischer, Gesellschaft für Mathematik und Datenverarbeitung mbH, Institut für Integrierte Publikations- und Informationssysteme (GMD-IPSI), Dolivostr. 15, D-6100 Darmstadt

Reports and Communications

Towards the Construction of a Thesaurus on the Italian Business History: An Announcement

Within the cultural program carried out by the Milan Chamber of Commerce in collaboration with the Foundation ASSI, work on the construction of a thesaurus on the Italian business history has begun. The team of 5 collaborators comprise R.BERTELLI and E.ROMANO from the Milan Chamber of Commerce, B.BEZZA and P.A.TONINELLI from the Fondazione ASSI in Milan and M.ARRIGONI, a Consultant in Library and Information Science.

When building a thesaurus on the Italian business history, the peculiarities of the economic growth in Italy, and consequently, of the organization of the entrepreneural, managerial and manufacturing activities are to be taken into account. Unlike Great Britain, the USA and Germany, where productive units have evolved following a sufficiently linear process (family-run units, joint-stock companies, vertical and/or horizontal integration, large-sized companies), in the past decades in Italy, traditionally "archaic" entrepreneural organizations (e.g. small-sized firms, family-run firms) have shown new vitality and coexist with more advanced forms of enterprises, such as public companies, or original forms, such as foreign-Italian holdings and state-owned companies. In the context of these peculiarities, all the organized forms of production, distribution and services which have characterized the Italian economy since the industrial revolution must be taken into account.

The thesaurus structure will be in conformity with the ISO 2788 Standard. The TINTERM software, developed by IME Ltd (London) has been selected for the automatic processing of terms and their relationships.

The selection of terms is being done by the Committee following literary warrant. Due to the complexity of business history and its still unformal settlement, being an "in fieri" discipline and at the center of a "crossroad" where economic history, economics, industrial and finance economy, business administration, sociology converge, the opportunity to adopt a faceted structure is under evaluation to better define the complex interdependence of "preferred" terms.

An "in vivo" test at the end of the work is foreseen to be carried out in a special section of the Milan Chamber of Commerce library dedicated to the Italian business history. M.Arrigoni (abridged)

Address: Dr.Mariagrazia Arrigoni, Camera di Commercio Industria, Artigianato eAgricoltura, Biblioteca, Via Meravigli 9/B, I-20123 Milano.