Ambiguities In Object Specifications In View Of Data Testing

Dieter Richter[1]

Technical Report TR-92-034[2]

June 1992

Abstract

Checking data only relying on their specification is of importance when using neutral or standardized object models. Ambiguities arise during the tests because of specifications leaving a certain degree of freedom to the implementation. Based on an experimental background the observations and reflections about the reasons are systematically presented. It turns out that the transition (or mapping) from a specification of an object to a physical instance (or data set) has to take into consideration when defining neutral models. This transition which often has been seen as a technical question of the implementation or as the internal (hided) feature of a system appears as a particular point of the concept besides the specification.

One crucial point is the instance handling with respect to assign and comparison operations. The mapping from a specification into a database can be realized in various manners which leads to interpretation defects when testing independently. Another point is the weak scope definition in specifications. Several ambiguities are caused by it. A very frequent reason of misunderstandings is the imprecise or wrong understanding of the different relations between objects, logical and physical instances. There are approaches for more clear specifications. The last point is the representation of failures or more generally of the state of instances. A concept based on multiple inheritance seems to increase the abstraction level of state specifications on the same level as the used specification language is of.

1. Physikalisch-Technische Bundesanstalt Berlin. Research partly done while at International Computer Science Institute (ICSI), Berkeley, California, as exchange visitor
2. Technical Report of ICSI

Table of Contents

1.0 Introduction

A central point dealing with in this paper are problems coming up when doing data tests independently from their originating process but only relying on the specification.

In this introductory part we start to show the evidence of the discussed problems by mentioning two typical application areas.

Then we continue with some general remarks about the background of doing tests independently. For this purpose the software engineering approach is briefly sketched to help to understand some fundamental differences between testing in the development environment and testing independently.

It follows the introduction of the problems of the test of data which are the particular subject of dealing with in this paper.

At last in the introductory part the problem of ambiguities is briefly and very generally evaluated and the further contents of the paper is overviewed.

1.1 Relevant Applications

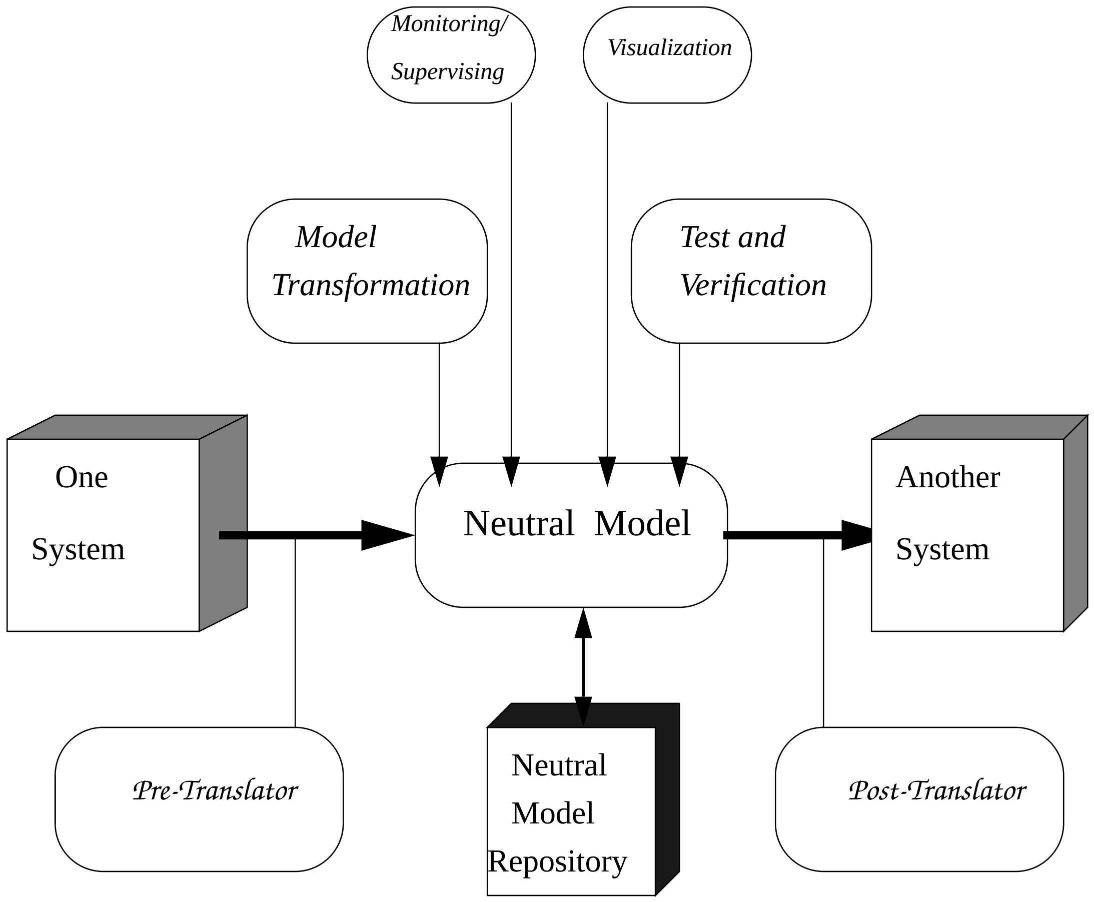
We outline two areas where the test of data with respect to a given specification plays an important role. In both cases there is an independent test necessary. In the first case the availability of the data generation environment is generally not to assume and in the second case for principle reasons it is to avoid to use the generation environment.

1.1.1 Data Exchange via Neutral Models

With the growing amount of data in computer integrated enterprises there has come up the need of exchanging data between different computer aided systems. The most sophisticated method of managing the increasing data exchange consists in the definition and implementation of so-called neutral models. These neutral models serve as intermediate formats for the exchange. They are the valid models during the exchange process. This method not only guarantees to need much less different software modules than direct connections between the different models of the systems do but it is a mean to implement the enterprise's central model as the ruling one. The arising problems are of general nature when information exchange becomes unified or even standardized.

An overview about the main functions necessary when exchanging information between different systems via neutral models is illustrated in the subsequent picture 1.

With the computer integrated manufacturing (CIM) the neutral model for the data exchange (the so-called neutral product data model) consists of entities to describe geometrical, technical, planning or other information structures which are relevant to design, plan or manufacture products.



Picture 1: Functions of data exchange via neutral models

The picture demonstrates the general case which is not only applicable in CIM. The information structure of the neutral model is described by applying a specification language. This language is expected to hold features to express the typical items of the desired application area. The design of the specification language should ideally follow two principles:

- to be expressible for the desired application area and usable by the normally non-specialist computer users
- to be unambiguous for the subsequent implementations.

But often this two requirements are contradictory and may not be satisfied simultaneously. The extreme cases are on the one hand when the natural language is used as the specification language (it had been used with the first model description for the data exchange between CAD systems) and on the other hand when a programming language is used for the specification even it is a high level, object oriented one.

A graphical support, diagrams or other user-friendly means integrated in the specification language increase the expressiveness in the application area but usually they imply less precise specifications with respect to the implementation and tests.

A question of intrinsic importance when exchanging data via neutral models is the check of correctness of data. This means it is to prove whether a data set after transferring fulfils the demanded model specification. There are a plenty of reasons why the specification could be violated.

One of them is to prove the result of any model transformation which is necessary to continue the data processing in another system. Another reason is to check the data before entering them in certain systems.

Basically, the neutral model turns out as a very convenient focus within the product data modelling where the correctness of exchanged data can be tested and supervised.

The test of data has to be done with respect to the specification of the neutral model. The general case is that one where the data already has been transferred from any system and eventually they has been transformed, too. It is not to expect that the tools used to define, create or transform the data are available with the neutral model. Moreover, it is to consider the case that it is unknown where the data are from. The only orientation for the test is the specification of the neutral model. This is a case we call independent test.

1.1.2 Intersystem Tests

A common way of proving, for instance, measurement instruments is the comparison of different ones. This comparison is performed by applying the instruments on same or comparable samples. In the case software is proved on this way the samples are data sets obeying same features in a certain sense.

Of special importance when performing intersystem tests is the assumption on the data being in all applications equally of the demanded form and structure. The certification of the test method has to include the prove of the correctness of data with respect to the demands on it, or with other words with respect to their specification.

An essential advantage of intersystem tests is the ability of comparing the different systems. The main prerequisites of a successful comparison are

- the reliability of the used sample data,
- the independence of performing the tests on the different systems.

The independence of each test on the different systems includes the independence of the correctness prove of the used data sets. The only criteria of correctness of the data is their correspondence to the given specification. Any knowledge from or tool of any system which underlies the intersystem test has to be excluded. Otherwise the independence and with it the comparability is lost or at least falsified.

1.2 Software Engineering and Testing

Specification languages or other means to describe information comfortably are of increasing use in many application areas. More and more graphical and other intelligent tools are developed to support the design of information structures. Usually these tools are integrated in software engineering tool boxes (CASE systems).

It is a goal of the specification tools to include as much as possible structures and knowledge from the application area they are designed for. Moreover it is intended that users from the specific application area without a deep knowledge of software technology do accept the tool. All this together leads to specification techniques turning to terms and structures of the application area and offering user-friendly standards for computational aspects. Actually this is not a new phenomena. It is the desired development on all levels of computer application.

In view of testing software there is a growing dependency on the engineering process. The correctness of a certain implementation can only be evaluated by regarding its technology of development. As the consequence the CASE tools provide support for all parts of the software lifecycle. Usually they are equipped with specification languages, graphical visualization tools, structure analyzers, code generators and not at least with test tools.

That is the situation very roughly outlined so far from the point of view of software technology.

Another view on software testing comes from the standardization of software interfaces. The unification and the following standardizations are necessary to communicate between different systems and to be able to replace a system without changing the environment. Standards have become an essential part of the information infrastructure like in the computer communication or in the CIM application area.

Usually standards consist of different parts. The actual contents are the descriptions of the information representing the interface. Often there are several layers, but this is not of importance for our considerations. To be unambiguous there are specification languages used to describe the contents of the interfaces. It is not unusual being developed a special specification languages for an interface description.

Unlike the software engineering with CASE we do not have here a full connected lifecycle. The use of specifications, the software development and the testing activities are going to be used at different places independently from each other.

Assumed a specification language is designed it may be used in various applications. For running applications it is to suppose that different software packages developed by different companies using different engineering technologies are available (indeed, that is the reason why a standardization is necessary). The role of testing consist here in proving whether an implementation claimed to perform a given specification does it correctly or not. The result of the test gets the rank of a certificate provided the testing institution fulfils the corresponding rights to make out certificates and the test has run successfully. It is obviously that tests whose purpose are certificates either must be designed independently from any used software engineering technology or must include the prove of the engineering technology itself.

Summarizing we notice two trends leaving with respect to the test technology a gap between them at least currently. The trends are:

- the CASE technology oriented to a connected software lifecycle,
- the standardization of an information infrastructure by introducing unified interfaces and models.

The gap is the ability of testing independently from the engineering environment. This problem has not been addressed sufficiently.

1.3 The Problem of Data Testing

What we have to do when testing a data set independently with respect to its specification is to check all the aspects of the specification being in force with the data set in question. This kind of checking presumes an existing mapping of the specification structures onto the structures of the implementation basis. At least it is assumed that programmer and tester have a common understanding about the mapping.

The implementations of data repositories use or follow usually in this circumstances data bases. Conceptional and structural frameworks base on the data management technology.

Against it, specification languages follow usually the philosophy of programming languages. The problem we have, that is to test whether a physical data set fulfills the requirement of the specification, corresponds to a certain extend to the determination whether the concepts of data base structures meets the intension of the corresponding language element or not. Generally this is not a surprisingly new issue. It has been approached by object oriented techniques.

But in our described circumstances the problem renders more difficult because there are to assume different tools used and there is an independence between programmer and tester due to the character of the data. The only common reference is the specification language. This leads to ambiguities and misunderstandings because of a certain degree of freedom left to the programmer. He has to decide depending on the implementation basis how to determine the open points of the specification. The consequence is the fact hat different programmers in different environments may decide in different manners. How then to decide what is a correct implementation? How to evaluate the results of tests?

By the way, a programmer’s decision of the above mentioned sense may be a decision of a single programmer using a certain implementation basis, but it may also be understood as the concept which is implemented by an implementation environment.

1.4 The Aim and the Character of the Paper

What we want to do in this paper is to identify which mapping problems exists. Our concern is neither the evaluation of different mappings nor the development of elements of an improved object oriented technique but the general revealing of transitional problems between specifications and physical repositories or data base. Often they has been mismatched with pure implementational ones. Mainly caused by the standardizations of neutral exchange models the transitional problems emerge as separated ones.

In this paper results and experiences are analyzed and interpreted systematically we got (see [8] through [13]) when developing test programs for checking data sets according to the neutral model of STEP. STEP is the informal name of an ISO project of standardizing a neutral product data model. There the specification language EXPRESS ([6]) is used which appears as a relatively strong and formal language. The fact that there several weak points with respect to an unambiguous specification from the view of independent testing has been found is of general significance. It permits to conclude

- these weak points are to expect repeatedly in application oriented specification,
- the solution will not be found by choosing or developing a convenient implementation technique,
- the standardization bodies should address the so-called transitional problems between specification and implementation, too.

We follow in several considerations the examples from STEP. But it is not of importance to have an inside into the STEP project or to know the EXPRESS language when following up the next sections. The used examples are self-evident. If there will be needed a special element from that background it will be explained.

The presentation is of a form collecting up in a systematic way what has been observed and found out about the nature of the problem based on experiences, discussions and the general knowledge of the technologies ([1] through [5]). It follows the basic principles about the techniques involved. It has been tried to avoid to discuss and analyze any special method of specification, implementation or testing. Even the effects of the special techniques used in the background of the mentioned project are generalized as much as possible.

1.5 Overview about the Following Sections

A very focussing point is the handling of instances created with respect to an object definition of a specification language. In section 2 we deal with the problems arising from different philosophies of instance identification and with the consequences on assign and comparison operations.

Section 3 is devoted to scope definitions. The difference between scopes founded on a semantic background and scopes on the management level is discussed and the problems of mappings between them are presented.

In Section 4 we unfold the different types of graphs of objects and instances. In our opinion they have led repeatedly to misunderstandings.

Section 5 deals with the state and error representation. It might be thought as a kind of representing the results of tests as well.

2.0 The Instance Handling

2.1 The General Problem

Basically, there is a difference of the understanding between that what an instance is on the specification level and that what it is on the data management level. We analyze at first this difference including the impact on the role of the instance identification.

2.1.1 Instances on the Specification Level

The most interesting point with high level languages is the definition of objects[1] or of object types, respectively. We identify these both notions as identical unless it is strictly mentioned another meaning.

What is an object? An object is the definition of a real thing by its aspects representing it. The aspects usually called attributes and are characterized by types, methods, rules or relation to other objects. In our considerations the main point of the object definition is the definition itself and the set up of relations to other objects.

An instance is a realization of an object. On this level it is not important for the understanding how the realization is performed. It is an intrinsic characteristic of instances that they are carriers of the properties of the object they are assigned to. Each instance of an object fulfills the properties defined by this object.

The instance identification plays the role of relating an instance to its object (type) and of discriminating logically different instances. The usually used method is the introduction of variables.

2.1.2 Instances on the Data Management Level

The key concept of data management systems is the data model. It depends on the kind of data base whether explicitly a data model is introduced and eventually which one the management system is based on. For the purpose of illustrations in this paper it is not important that the data model is explicitly given. If not let us assume there does exist implicitly a data model. The main point of the data model is to support certain management functions like access or manipulation.

1. We include as well the class definition in object orient languages but the term ‘object’ is constantly used in this paper, see also paragraph 2.1.3.

On the management level an instance is a physically existing data set according to the data model. The instance fulfills the characteristics of the data model which in a great deal are structural relations.

The instance identification plays the role of identifying a data set and of making it accessible by the management system. Usually there are used user given names or surrogates are introduced by the management system.

2.1.3 The Transitional Problems

When mapping instances of the specification level onto instances of the management level then there a couple of decisions are to make. This is necessary due to the different concepts. Usually not all the features covered and required by the data management system are deductible from the specification level. But which ones are deductible and which ones not? The opinions can be divided about that. The programmer realizes his opinion. But even this is the problem of testing the implementation independently. The tester might have another opinion without being aware that other interpretations are allowed, too.

Vice versa, if from the management level an access is necessary to a type or generally to an object definition then it lacks mostly the relation to it. In a concrete case it is then a question of chance or of skill to find a technical solution. This is actually not a problem of ambiguity but it is also related to an insufficient transferability between concepts of specification and of data management.

The existence of these problems is neither the outcome of a bad specification language or its usage nor the consequence of applying low level data bases. It represents the questions concerning the instance testing when developing and performing data exchange via neutral models or more generally when testing independently from their creation.

With object oriented techniques classes are introduced incorporating the creation method of instances into the specification of objects. This is exactly what is needed in each language for an unambiguous transition from the specification to the data repository. The unambiguity is reached for a single implementation. But in each implementation exists a dependency on the used tools and basis. So there are different realizations coming out causing the same problems as described. Therefore we do not further pursue the class concept. The term object is used through this paper knowing that as well a class definition or a any type definition may be meant.

What we want to do is to indicate transitional ambiguities and to emphasize their consideration.

What are finally the problems we found? We list them without discussing in detail here, each of them is devoted a particular section:

- Different variants to identify instances of the management level (problem of uniqueness of mapping from specification to management)

- Recognition of the corresponding object type of an given instance on the management level (problem of reversal mapping from management to specification)
- Realization of assign operations (problem of uniqueness of mapping from specification to management)
- Realization of Comparisons (problem of uniqueness of mapping from specification to management)
- How to decide about the persistence of data on the management level (problem of existence of mapping from specification to management)

2.2 Particular Points of Instance Handling

The subsequent points are those we recovered when developing instance tests. For illustration reasons they has been refined and arranged in a certain systematical order not necessarily directly corresponding to their real appearance. Of course, there is no claim to be complete.

2.2.1 Different Instance Identifications on the Management Level

The problem does not consist in using different names in different systems for the same instance. The actual problem is that by naming an information concerning the contents may be placed. The two major forms of doing this are

- the placement of identifiers having a semantic relation to the contents and
- the introduction of a kind of order by the identifiers.

On the specification level this kind of naming is usual and often realized by any systems. It increases the readability of specifications. So far this principle is useful.

But with respect to the independent data test there is a danger. It comes into force if the identifiers contain an information which is otherwise used by the system than to serve for a better readability and if simultaneously this usage of naming is not a part of the specification language. What then happens is a maintaining of the introduced principle of identifiers when storing the instances into the local data base. This is not a problem for the local systems where the instances has been created. But it turns out to be a problem when transferring the instances to another system or to a neutral model repository. Then the relation to the former system environment has been cut and the additional information carried by the identifiers is worthless. Any structure a set of instances might be have and not corresponding to a specification item is subject to loss.

We have described how different instance identifications can arise. The problem we want to point out is that the differences are caused by system specific naming principles. It becomes dangerous when the naming principle is not covered by the specification language. Then the background and with it the explanation of the principles is lost when the instances are moved from the system they has been created.

Unfortunately, the phenomena described is not seldom. It is even understandable from the application’s point of view. Thinking in terms of the application area it is difficult to restrict strongly to the formal points given by the specification language.

An example should demonstrate how fast you might be tempted to introduce an identification concerning the contents not covered by the specification language. We use EXPRESS ([6]) for this example. The application is arbitrary and not encouraged by the language. But it is typical for the problem we are discussing about.

Example 1[1]:
Within geometrical applications ([7]) of EXPRESS there are introduced different kinds of point definitions. Each kind is defined as an object. An object is called entity generally in EXPRESS. We are interested now in two of the different point entities: the point_on_curve entity and the point_on_surface entity. We give the definition as it is standardized for them. Capitalized terms are predefined in EXPRESS.

```
ENTITY point_on_curve
SUBTYPE OF (point);
  basis_curve       : curve;
  point_parameter: REAL;
DERIVE
  dim: INTEGER:= coordinate_space(basis_curve);
WHERE
  WR1: {parametric_lower_limit (basis_curve) <= point_parameter <=
                                        parametric_upper_limit (basis_curve)};
END ENTITY;

ENTITY point_on_surface
SUBTYPE OF (point);
  basis_curve       : surface;
  point_parameter_u: REAL;
  point_parameter_v: REAL;
DERIVE
  dim: INTEGER:= coordinate_space(basis_surface);
WHERE
  WR1:  {parametric_lower_limit (basis_surface)[1] <= point_parameter_u <=
                                   parametric_upper_limit (basis_surface)[1]} AND
        {parametric_lower_limit (basis_surface)[2] <= point_parameter_v <=
                                   parametric_upper_limit (basis_surface)[2]};
END ENTITY;
```

The completely presented definitions are self-evident or, where not, of no interest here. Instances of these entities let us assume be named in a certain environment either ‘point_n’ or ‘point_n_m’ depending on

1. In the first examples it is necessary to introduce one or another language element. If it will be used in latter examples, too, it will not be explained repeatedly

belonging to the first or to the second entity, respectively. n and m are representing counters of integer type. The two different point entities might be used to characterize either points on the boundary or on the interior of a surface, respectively. The temptation to use the knowledge which is contained in the identifiers is present. For instance, when a test procedure uses the knowledge about naming to select an appropriate access method to the attributes of the instances. Of course, this access method must fail, if the supposed naming convention is not valid.

This example might be obvious. We used it to demonstrate the kernel of the problem. In real applications it is mostly more difficult to recognize the root of the problem.

2.2.2 Recognition of the Corresponding Object Type

The determination of the object type to a given physical instance[1] turns out as a function of highest importance. Many of the implemented methods are dedicated to special types. By introducing object hierarchies and using the inheritance it becomes necessary to determine the special subtype in order to select the appropriate method. With respect to independent data tests the importance of type determinations is emphasized. As a typical situation appears the case that a set of physical instances has been transferred to a neutral model repository. Before transferring them to another system there are modifications and tests necessary. The specification is available for the test, but often there is no information about the degree of specialization a concrete instance is subjected. Instead of this it is to put up with a more general specification known as surely valid for that instance. However, this is a situation not sufficient doing the necessary tests.

Necessary is a technique which allows to determine the exact type when having the instance alone. Most of the known programming systems do not support the reversal mapping from the physical instance (instance on the management level) to the object definition on the specification level. So it is with EXPRESS and C^{++}. An exception is the language SATHER ([5]) which automatically introduces an attribute named ‘type’. This attribute establishes the desired relation from the physical instance to the object definition.

But in view from the independent data test it is not sufficient to demand any support for the reversal mapping to the type. It is a convention necessary to establish this mapping uniquely. Otherwise the type recognition can not be ensured.

2.2.3 Realization of Assign Operations

The very simple question is how to realize an assign operation of the kind

a := b

given on the specification level physically on the data management level. Again, the problem is not to have any answer. Of course, there are solution with each compiler. The point is that there are different realizations applicable. As variants we have

1. Physical instances is a synonymous term of instance on the data management level.

- The variable a (a and b appear here as the identifiers of instances of the specification level) gets related to the same physical instance as b already is.
- An existing physical instance related to a is overwritten by all the attribute values of the physical instance currently related to b.
- A new physical instance is created and related to a. All the attributes get the values of the corresponding attributes of the physical instance currently related to b.
- If an attribute is a reference to another physical instance then this other instance is also copied or a new one is created for, respectively (this is an alternative to the second or the third variant, respectively).

Concerning the last point high level languages introduce variables of pointer types in order to be able to distinguish between the value and the address of instances. But even this is no unambiguous instruction how to realize the assign operation physically with the data management. There are other influences on the data management like time and space saving. Instances can become very complex so that time and space are real factors. Particularly when the background of the instance creation is unknown it might be rather difficult to understand the reference structure alone out of the specification.

The problem described above should be evident without providing any detailed example. We were confronted with this problem when checking the identity of physical instances specified by EXPRESS. In practical applications it is often important to distinguish between “two entities establish as the same one” and “two entities are equal”. It makes the difference whether, for instance, two references to a part of a machine mean the same one or equal ones[1]. Backward concluding we had to notice that the necessary prerequisite, that means, the distinction of different kinds of assign operations was not provided.

2.2.4 Realizations of Comparisons

Similarly, the realization of comparisons raises nearly the same questions as the assign operation does. The question is: What is the physical interpretation of a comparison like

$$a = b$$

given on the specification level? The following variants exist:

- a and b are referencing to the same physical instance.
- All the attributes of the physical instances referenced by a and b are equal.
- The attributes of the physical instances referenced by a and b are equal, except the attribute is a reference to another instance. Then the attributes of those corresponding instances are equal.

As an evident conclusion there is needed a distinction between “same” and “equal” physical instances when comparing instances on the specification level. However, an intrinsic

1. This point is discussed in the next paragraph, too.

prerequisite to realize different kinds of comparisons is that even this distinction is realized with assign operations. It concerns not only the direct assignment (:=), but also all other kinds like, for instance, the function calls.

The argumentation and the background for problems arising with comparisons are likely the same as with assign operation (see the paragraph before). In EXPRESS there are introduced two different comparison operations, namely

a = b

and

a :=: b

The first case means the above mentioned second or third interpretation, it is not specified more precisely, and the second case means the first interpretation. However, there is introduced only one assign operation (:=) so that the different comparison operators cannot be taken advantage of.

2.2.5 The Persistence of Data

When should the physical realization of an instance created on the specification level be persistent?

This is the question to be discussed in this paragraph. However, this question is hardly to answer in a general manner as we will argue below. The semantic purpose of creating instances determines much more the corresponding solution than it does concerning the points presented in the paragraphs before.

The persistence or on the opposite the transiency of data is a basic feature with respect to data base management. On this level it is a feature which implies the use of permanent storage capabilities or not, respectively.

But on the specification level there is usually not any concept supporting directly the persistence specification. What are possibly similar concepts on this level applicable to persistence characterization?

There are two concepts being worth to discuss in this circumstance.

The first one is the local definition of variables in functions or procedures. One might be inclined to relate the local variables to the transiency and the global ones to the persistence. But is it well sounded when locally defined variables are mapped onto transient data sets and globally defined ones are mapped onto persistent data sets? Though this mapping is the only one making partly sense it does not correctly interpret the meaning of a local variable. There is not reasonably to conclude from the definition range of variables to the life durability of data. Another point which contradicts the approach is the hierarchical structure of programs resulting in variables which are global with respect to one and local with respect to another procedure.

The second concept is a generalization of the first one. It concerns the visibility concept of object oriented languages. Is the visibility of variables an expression of the persistence of data assigned to? And vice versa, can the invisibility be mapped to the transiency? Surely, they can not. The argumentation is likely the above one.

The reason of the unsatisfied situation is the lack of suitable data description elements in application specification languages. From the current point of view for the reason of user acceptance it is not to recommend to enrich those languages by data base oriented description elements. Furthermore, there are standards necessary bridging the gap and being mandatorily to apply. One very simple approach is the introduction of name conventions in an application area indicating the persistence or transiency, respectively.

3.0 Scope Definitions

3.1 The General Problem

The term scope is used in different applications to identify a whole area which it deals with or includes. The understanding of scopes in application areas differs as well each from others as from structural domain definitions on the management level. We analyze at first the differences generally.

3.1.1 Scopes on the Specification Level[1]

On the semantic level a scope is an area of objects put together by semantic relations. The main points of the scope definition are

- the semantic definition of the area considered,
- the demarcation of this area.

Usually the semantic scopes are defined by the semantic relations of the corresponding application area. Often it is an informal representation based on the knowledge of the area.

Two examples should illustrate the introduction of scopes.

Example 2: The Scope of Positive Definite Matrices

In many numerical applications there is of interest a class of matrices being positive definite. For the example we assume there is to perform the inversion of matrices. In any case, if a numerical framework is applied then the term ‘positive definite’ need not to be explained. Otherwise it is to give a description what features are representing this property and what methods are permit to apply. What is the scope introduced?

It consists of all matrices of arbitrary dimension being positive definite. Additionally the methods to invert this type of matrices are belonging to the scope too. Furthermore there are rules and theorems valid for positive definite matrices which are to include in the scope.

1. Below this scopes are synonymously called semantic scopes.

Consequently the scope represents the knowledge about an area which is necessary or useful, respectively, to apply.

Example 3: The Schema of Geometric and Topological Representation in STEP

In EXPRESS an scheme concept is introduced which appears as a closure around a bunch of entities belonging together by any semantic background. The application of the scheme concept is not restricted anyway. This concept is introduced formally to specify scopes. In the case of this example the scope consists of all the entities and associated data types and constraint functions used in the explicit representation of the shape of technical model.

The introduced formal form of the scheme definition is

```
 SCHEMA geometry_scheme
 (* arbitrary specifications*)
END_SCHEME;
```

In one of the subsequent paragraphs we will present an example where this formal scheme definition is used for references between schemes.

With both examples we can see that the definition of scopes depends fully on the given or intended semantic background. There is no difference when a formal description of scopes is introduced as the example 3 shows.

3.1.2 Scopes on the Management Level

The purpose of scopes on the management level consists in supporting management functions by gathering up the data sets. The definition depends on the concept of the data base.

The variants range from file, record or segment definitions in more poor data management systems up to class definitions in object oriented systems. Generally, it realizes the applicability of an management function to the whole scope.

3.1.3 The Mapping Problem

Having a scope on the specification level it is desired to maintain this scope information on the physical level. There are several problems when realizing this requirement. Two major ones are discussed later in detail.

The general problem is that each different implementation environment implies a different mapping. That means there is structural information to expect which can not be interpreted independently from the origin. Especially, it is not possible to distinguish between structures introduced by the semantic scope and those physically arisen by the used data base. When transferring such data to neutral models misinterpretations are to foresee.

As an illustration we present an example.

Example 4:

We use one of the point definitions of the example 1, let us take the point_on_curve. Assumed there are given a set of curves and a set of points defined by using this curves. Based on the set of points be defined a set of lines by applying the following specification.

```
ENTITY line
  point_1: point_on_curve;
  point_2: point_on_curve;
WHERE
  WR1: point_1 <> point_2;
END_ENTITY;
```

Furthermore we assume as a data base there is used a file management system with a variable file length. For performance reasons be one file contained of one curve entity, all point entities lying on this curve and those line entities whose attribute point_1 lies on this curve.

We get by this storage concept a quasistructure of lines with respect to the basic curves. This structure is not covered by the specification. But it might become rather hard to recognize, when the set of lines is transferred to a neutral model or to another system possibly portioned according to this quasistructure.

3.2 Handling Global Rules

There are different types of rules used in specifications. They differ by applying to one object (type) or to a scope of objects. Before discussing the problems of global rules we introduce at first the both types of rules. The terms might not be in common use.

3.2.1 Local Rules

Local rules are constraints that apply for each single instance of an object. They are part of the object definition on the specification level. The local rules are valid for the instances independently from each other. The representation of these rules on the management level is realized by data sets fulfilling them. There is no other mapping and no relation between the application of the rules to different instances.

When proving the correctness of an instance with respect to the specification then the fulfillment of the rule has to be a part of the testing process.

The kind of rules can vary. There are simple restrictions of data ranges possible as well as complex functional constraints. We cite two examples from STEP (Part 42: Geometry and Topology, [7]). There the local rules are specified by WHERE clauses.

Example 5: Circle Definition

This is an example for a simple data restriction. The entities used as reference but not introduced are not of interest in detail. Their meaning is obvious by their names.

```
ENTITY circle
SUBTYPE OF (conic);
  radius   : length_measure;
  position: axis2_placement;
DERIVE
  dim      : INTEGER:= coordinate_space(position);
WHERE
  WR1: radius > 0.0;
END_ENTITY;
```

Example 6: Offset Curve Definition

This example exhibits the use of a function within the rule. Again, the entities and types not introduced are obviously understandable for the context here needed.

```
ENTITY d2_offset_curve
SUBTYPE OF (offset_curve);
  basis_curve   : curve;
  distance:       : length_measure;
  self_intersect: OPTIONAL BOOLEAN;
DERIVE
  dim           : INTEGER:= coordinate_space(basis_curve);
WHERE
  WR1: dim = 2;
END_ENTITY;
```

What here looks like a simple restriction of an attribute is actually the application of the function coordinate_space which determines the dimension of the space spanned by the curve.

3.2.2 Global Rules

Global rules are those ones taking into account simultaneously more than one object or more than one instance of an object or both of them. Actually, there should be introduced different types of global rules differing the mentioned variants. But we disregard in this paper.

The consideration of scopes comes along with the handling of global rules. In the case that there are no designated objects concerned in a global rule another range definition must be existing. This role is expected to be played by the scope concept. Whether it does will be discussed in the next paragraph.

Global rules may be constraints applicable to all the single instances of certain objects but need not. It depends on the type of the rule. If the rule is an extension of a local role by including into the constraints more than one object then the similar principle holds as with local rules. This means the rules are valid for all instance combinations of the concerned objects. If, for instance, the rule is a limitation of the quantity of instances then the rule is not to apply on a single instance or an instance combination, respectively. In this case it is an application on the whole set of instances concerned by the rule. Especially when establishing a limitation of quantity then the domain must be specified. Sometimes this domain is specified by relating to one or several determined objects. If the object is not the basis for the quantity measure then another frame is necessary. One of the most used quantity limitation is the uniqueness of instances.

EXPRESS introduces the scheme concept as already mentioned. In addition to the WHERE clauses there is the RULE concept serving as frame for global rules. For the special case of uniqueness constraints there is introduced the UNIQUE clause. We give two examples to demonstrate the diversity of global rules.

Example 7: Unique Rule

If an object has several attributes then each individual one may be constraint by a uniqueness rule as well as a combination of them. This means we can have (in EXPRESS notification)

```
ENTITY person_name
  last      : STRING;
  first     : STRING;
  middle    : STRING;
  nickname: STRING;
UNIQUE
  nickname: STRING;
END_ENTITY;
```

where it is demanded that in the present physical scope each nickname must not occur twice. Another variant is

```
ENTITY employee;
  badge: NUMBER;
  name : person_name[1];
UNIQUE
  badge, name;
END_ENTITY;
```

where each combination of name and badge must be unique within the physical scope.

Example 8: point and line match

This example exhibits a rule which is a combination of a quantity restriction and a functional relation between different entities. We assume as defined a point and a line entity. function_1 be a relation between a point and a line with a boolean result. Then the following rule can be defined.

```
RULE point_and_line_match FOR (point, line);
LOCAL
  p: point;
  l: line;
  s: SET OF BOOLEAN;
END_LOCAL;
  s:= QUERY(function_1(p,l) | function_1(p,l) = TRUE);
WHERE
  SIZEOF(s) <= 10;
END_RULE;
```

The rule says that there are not more than 10 tupels each consisting of one point and one line are allowed fulfilling the relation function_1.

3.2.3 Mapping and Testing the Rules

Whereas local rules are rather clearly to prove there are problems with global ones. The difficulties are caused by the ambiguous transition from the specification to the data man-

1. As person_name the above definition is assumed without the unique clause for the nickname.

agement level with respect to the scope. A test program for global rules depends on the implementation of the semantic global rules. This implementation again depends on the physical scope concept available with the implementation basis.

Coming back to our initial approach of testing instances of a neutral model there are two major problems concerning global rules.

The first one concerns the consequences of using different and for the data test generally unknown physical scope concepts in systems where the instances has been created. The transition from the logical scope to the data set oriented scope leaves a great degree of freedom. One realization can be quiet different from another. There are structures of data sets possible which are not to decide about whether they are representing a logical scope or are arisen for technical reasons. For instances, assumed a sequence of scopes is realized according to the same specification and each of the scopes consist of such a large number of instances that with respect to the used data base system it has been split into several files for technical reasons. Let us further assume all the instances of the whole set of files is transferred to a neutral repository. This scenario just described is a very realistic one. We met it repeatedly. It is then a nontrivial task to recognize which of the files are belonging together and forming one scope. The specification does not provide any information about the splitting! Each testing procedure would be dependent on information about the origin of the instance creation. This is not realizable in each case and is a source of misunderstandings and misleading test results.

The second problem is the insufficient scope concept on the specification level itself. It seems to be not appropriate to be used as a frame for the validity range definition of global rules. The semantic variety of validity ranges requires actually a much more subdivided structure of scopes as it is with the applications we know. Doing this it would result in an additional structure above the level of objects which needs again a concept to manage it. The class concept of object oriented languages seems not to be a concept replacing usefully the discussed scopes. There is a flexibility like the addition, change or cancelation of rules needed which is not supported be classes. We recommend to avoid global rules generally. There is a concept which makes it only with local rules and quantity restrictions relating to instances of one object. What necessarily is to do is to introduce instead of scopes new objects, gathering all those objects a global rule concerns. The idea is very similar to the concept we describe more in detail in the section 5.2.2 when we discuss the state representation problem and propose a concept.

3.3 Using Foreign Object Definitions

Once there is existing a scope structure including the objects two kinds of using predefined objects are occurring with a new scope to be defined.

- It is desired to include an already existing object into the new scope[1]. An important point is that then at least two scopes contain the same object.

1. It is not only a copy of the text defining the existing object meant but a inclusion of an existing object.

- There is a reference of an object attribute to an object not belonging to the current scope. To resolve the reference it should be allowed to use the corresponding object definition without integrating it into the current scope.

The first point causes already some conceptional contradictions. The consequence of it is having one object and all its instances in two different scopes. This again is not compatible to the validity range of instance identifiers and contradicts to the principle of global rules. However, there are languages having a similar concept as mentioned in the first point: It is the concept of COMMON blocks in FORTRAN and the option SHARED for attributes in SATHER. It is for many applications an useful and favorite concept. Nevertheless, concerning the ambiguity of implemented instances there are some unanswered questions like

- How learns a scope which shares an object with another scope about the creation of an instance of the shared object?
- How to transfer the instance identification from the scope where it is created to another scope where its object definition is valid, too?
- How to avoid same identifiers of instance of shared objects in different scopes?
- How to apply the uniqueness rule or other global rules onto instances of shared objects?

These questions are reason enough strongly to recommend to avoid the inclusion of objects. Based on these conceptional ambiguities no reasonable and testable realization is possible.

Concerning the reference to objects in another scopes it behaves less difficult as in the first case with respect to testable realizations. When in a scope an instance is created, then the reference to another object is called in to know the specification for this particular part of the object. It is not generated an instance in the foreign scope. The same way a testing procedure can operate. Concludingly the reference to foreign objects causes no new problems provided the sketched philosophy is commonly accepted and followed.

There is a question arising more generally with attributes which are references to another objects. The point is whether an instance creation implies a creation of an instance of the object which is referenced to, too, or not. The same question can be posed as whether from existing instances belonging to objects which are referenced to by another object can be composed an instance of the referencing object without generating new data. We shall discuss this problem more deeply in the paragraph 4.1.3. of the next section.

4.0 Graphs of Objects and Instance Relations

The reason this chapter has been included into the paper is motivated by a repeatedly observed mismatch of the different relation levels. At first we represent the relations and discuss the emerging questions. In the second part of this chapter we deal with some interrelated problems between different relations.

4.1 The Relations

We follow the introduced concept of objects and instances on the specification level and of instances on the data management level. Based on it all the known and theoretically possible relations are presented.

4.1.1 The Graph of Referenced Objects

On the specification level it is a very common practice to define an attribute of an object by a reference to another object. This kind of specification expresses not more and not less that the corresponding attribute is of the same nature as the whole object is which is referenced to.

This reference between objects does not say anything about the existence of instances of the concerned objects. One of the most met mistakes is the attempt to conclude a statement about even this point out of the object references. It is quiet clear that there must be installed a compatible relation between the object references and any used instance reference. But this relation is not a matter of the object references and can not be derived from it (see also the paragraphs 4.1.3 and 4.2.1). From the point of view of testing instances independently any conclusion of this kind without any additional definition must fail because there is no unambiguity.

Because this type of references between objects (or types) is very common and used in all the known high level and object oriented languages there is no illustration by an example necessary.

In most of the languages with object or type references there is a recursive use of the reference not allowed. Though this is not the point to be discussed in this paper we exclude recursive references for simplicity reasons.

4.1.2 Hierarchy of Objects

By the hierarchy of objects we refer to that what is better known as the hierarchy of classes in object oriented languages. For keeping the same notation through this paper we use the term as introduced in the headline of this paragraph.

With object oriented languages there is introduced the concept of inheritance. This means, an object may take over (inherit) an attribute of another object and regard this attribute as declared by itself.

The inheritance is a very powerful specification mean. It is used to specify which of the attributes are identical among a set of objects and leads to a hierachical relation between this objects (subobjects) and a so-called superobject from which they inherit. All attributes defined in the superobject are automatically defined for all the subobjects belonging to it. Subobjects represent a specialization of the superobject or, vice versa, superobjects are a generalization of the subobjects.

We give an example.

Example 9: Curve as Superentity

We take this example again from the specification of geometry by EXPRESS in STEP. There a superentity curve is defined with six subentities.

```
ENTITY curve
SUBTYPE OF (geometry);
WHERE
  WR1: arcwise_connected (SELF);
  WR2: arc_length_extend (SELF) > 0;
END_ENTITY;
```

The entity is already a subentity of another one, therefore it inherits the attributes of the entity geometry (in the concrete case geometry has no attribute). The two local rules defined in the entity curve are valid for all the subentities (they restrict the curves to be connected and of positive length). We give one of the subentities.

```
ENTITY line
SUBTYPE OF (curve);
  pnt: cartesian_point;
  dir: direction;
DERIVE
  dim: INTEGER:= coordinate_space(pnt);
WHERE
  WR1: coordinate_space(pnt) = coordinate_space(dir);
END_ENTITY;
```

The hierarchical relation based on the inheritance is again a relation among the objects on the specification level. But on the contrary to the reference relation in this case there is an implication on the instantiation: Each instance of a subobject is automatically an instance of the corresponding superobject, but, of course, not vice versa. This implication follows from the logic of the specialization and generalization, respectively. Each element of a specialized class must be an element of the more general class.

4.1.3 The Graph of Referenced Instances on the Specification Level.

Let us at first introduce a relation among instances on the specification level and then discuss it. This type of relation is realized in languages where pointer variables are known. We remind the interpretation of instances on the specification level. They are carriers of the object definitions and the kind of realization is of no importance.

In high level languages a variable which is not of pointer type carries the attribute values independently from other variables even in the case there is a reference relation between the types defined. In order to be able to exploit a defined reference relation of the objects in variables (instances)[1], too, the so-called pointer variables are introduced. When an

1. The terms variable and instance are used synonymously.

attribute of an object is a reference to another object and the corresponding attribute of an assigned variable is a pointer type then this instance does not copy or integrate an instance of the referenced object but uses the already existing instance by pointing to it. Usually pointer variables are of a certain type (object), that means they can point only on variables of this type (or on instances of this object).Then a value of the pointer variable is the identification of a determined variable (or instance).

The principle of pointer variables establishes a reference system among instances. But in order to apply these pointers a basic prerequisite must be given: On the specification level has to be introduced the so-called pointer types. That means, with respect to references to other objects already when defining objects on the specification level it is to distinguish between attributes which are of the pointer type and ones which are fully include an instance of the referenced object. From the application's point of view this is impracticable because it is generally not important for the semantic aspect of the application. Furthermore, it is inflexible already at this specification time to decide about detailed process realization aspect. This decision should be left up to more specific algorithmic or process definitions or even up to the realization when surely different variants of realizations for the same specification depending on the concrete purpose are imaginable.

But how to come to unambiguous specifications from the view of the implementation and testing? The above argumentation seems to be in contradiction to the aim of determining unambiguous implementations of neutral models, but it seems only. We do not plead for weaker application specifications. What we want to express is the difference between that what a matter of a object specification on the one hand and of the instance definition on the other hand is. Provided this distinction is realized the so-called transition to physical instances is a matter of mapping only from the logical instances to physical ones. The object definition on the specification level would not be affected by the transition.

In the case of neutral models both aspects, the object definition and the instance definition need to become a matter of standardization. In the case of instance references we now discuss the transitional ambiguity has been solved till now, if solved, however, by drawing up the problem onto the object specification where it is misplaced as outlined.

From the kind of the instantiation on the specification level must be derivable the kind of physical instances on the data management level. These is the only way to come to unambiguous specifications from the point of view of implementing and testing. The instantiation on the specification level can either be done by application specification designer or can also be predefined by tools used on the specification level.

The following example is a theoretical one to demonstrate one variant of a better distinction between the role of object definitions and the one of instantiations.

Example 10: Referenced Instances on the Specification Level (PASCAL-like)

We introduce at first a type which is later referenced to[1].

atype = record

1. All the introduced types can be thought as objects as well and the variables as instances, too.

```
  attribute_a1: type_1;
  attribute_a2: type_2;
 ...
end;
```

In current high level languages we can establish two different kind of references. The first one, called btype, takes the referenced type atype to define a certain attribute. Each variable declared for btype does not have any relation to variables of the type atype.

```
btype = record
  attribute_b1: atype;
  attribute_b2: any_type;
 ...
end;
```

The second one, called ctype, establishes a relation to the type atype by introducing a pointer type. This pointer means that each variable of ctype does not have any realization for the attribute defined by atype but relates to any variable of atype.

```
ctype = record
  attribute_c1: ^atype;
  attribute_c2: any_type;
 ...
end;
```

In order to avoid the decision among the variants at the type definition time we propose a concept as sketched by example as follows.

The concept uses only one variant to establish references in type definitions but differs two assign forms. As a matter of principle it is used only the first of the above variants when defining types, that means we assume the type btype to be defined. Then let us further assume there are variables declared as follows:

```
  a1, a2: atype;
  b       : btype;
```

Now there can be introduce two kinds of assign operation. The first one as

```
  b:= any_valid_expression;
```

or

```
  b.attribute_b1:= any_valid_expression;
```

or

```
  b.attribute_b1:= a1;
```

is to understand as it is usually used. The attributes of the variable b get their attributes according to the types defined and after realizing the assign operation there is no relation to the variables of the type atype. In the second case we propose a new kind of assign operations. By

```
  b.a1:= any_valid_expression;
```

or

```
  b.a1:= a2;
```

we explicitly relate the first attribute of the variable b to the variable a1 of the type atype. Instead of assigning values to the attribute attribute_b1 as in the first case we point in the second case to the determined variable a1 of the type atype.

It is evident that by the proposed concept the two most desired points are fulfilled:

- The object definition is independent from the instance definition.
- The instance mapping characteristics known from the pointer concept are kept.

However, we must concede, the realization of a concept as proposed in the example has not been proved as realizable so far.

4.1.4 The Graph of Referenced Instances on the Management Level

This type of relation between physical instances is a very well known concept of data bases. It says an instance can contain as data a reference to another instance. Depending on the data base this reference is an address, a symbolic reference or something like that.

The usual way to map referenced instances from the specification to the management level is to replace the symbolic relation between instances (variables) by data base oriented relations between the corresponding data sets.

One point which could lead to misunderstandings arises if the same kind of physical references is used as well for the image of relations from the specification level as for references being necessary to link data sets which has to be plucked to pieces for technical reasons.

4.2 Interrelated Problems

4.2.1 Transition from the Graph of Objects on the Specification Level to the Graph of Instances on the Management Level

When testing data with respect to the specification of a neutral model then there is given on the one hand the specification and on the other hand the physical instances. How the transition from the one to the other level has been performed is hided or is generally to be assumed as not available. This is the situation when testing physical instances independently.

In general the mapping from the object definition including object references to physical instances including their reference structure is possible on various ways. Which one is realized depends on several factors ranging from the specification language via the implementation environment up to the data base. The mapping is not reproducible when only the object structure and the physical structure is given. There is even no concept available which could guide and support the mappings at implementation time and based on this use serve as recognizing aid at testing time.

An approach we suggest consist in explicitly using the concept of referenced instances on the specification level as described in the paragraph 4.1.3. The advantage is the splitting of the mapping process into two ones:

- The first one maps the object reference structure onto an instance reference structure. Essentially it is based on the semantic analysis of the object structure with respect to the particular purpose of the concrete task. The simplest instance reference structure is that one having no relation between the instances. When using the terminology of PASCAL then there does not exist any pointer variable. On the opposite the most complex instance structure is realized when each reference between objects corresponds exclusively to relations between variables. In the case that to an object more than one instance is declared then one object relation can result in a set of instance relations. One example: According to the above presented example 10 to the existing object relation between the types atype and btype there could be introduce instance relations like
  b.a1:= any_valid_expression;
  b.a2:= any_valid_expression;
  in different assign statements. We use here the kind of relating instances introduced in the example 10 as a suggestion.
- The second one is the mapping from the instance reference structure onto a physical instance structure. This mapping has to manage the physical realization of an existing logical structure. Problems arising from the data base level are to tackle on this level. The splitting does not support problems of this kind. But the splitting sets the second mapping free from semantic interpretations of the specifications.

Though this approach does not cover all the mentioned problems of ambiguity it appears as promising with respect to the independent test of physical instances.

4.2.2 The Transition from Object Hierarchies to Physical Instance Structures

As already mentioned in the paragraph 4.1.2 there does exist one interrelation between object hierarchies and physical instance structures. It is derived from the general understanding of the specification or generalization, respectively. This interrelation is defined so far without any ambiguities. Technical problems coming up with the realization of instance sets and their inclusion relations are existing and affecting the transitional problems, but they are not existing due to this interrelation.

There are two other points we want to deal with. They are in one case resulting from implementational limitations and in the other case from an insufficient understanding of the specialization/generalization concept. In both cases they are not a consequence of the interrelation discussed but they are settled around and concern it.

The first point deals with the most frequently used instantiation of objects. Because of lacking a concept for realizing instances of superobjects they are not allowed as instances. There are only allowed instances of so-called objects of the leaf type, that means of the bottom-most objects according to the inheritance hierarchy. All the inherited attributes of all levels must be taken over explicitly by the instances of the leaf objects. The problem existing here is twofold. On the one side there must be a fully specification of all instances, there are not instances of superobjects realizable and on the other side by the repeated inclusion of attributes of superobjects the needed storage space can increase in a huge amount. Not regarding the mentioned restriction mapping problems in the actual sense of this paper does not appear by this point.

The second point concerns attempts to interpret more information in the specialization/generalization concept then it contains as it has been done, for instance, with the EXPRESS language. Before falling in this we want clearly point out what a specialization is.

A specialization is a logical subdivision of a set of elements according to certain defined features. A subset represents one specialization if all its elements have the same feature. With respect to the object definition it means a subobject distinguishes from others by having its special attributes. To each subobject belongs a set of instances. Regarding the sets of instances to all subobjects they are forming a non-overlapping division of the union of instances to all subobjects. With other words there is an ‘exclusive or’ relation existing concerning the containment of instances in the subobject related instance sets.

What is tried to do sometimes is to introduce other logical relations than the ‘exclusive or’ between subobjects. One of them is the simple ‘or’ relation which includes the ‘and’. What does it mean? It means it should be allowed that an instance is belonging to one subobject as well as to another subobject. But this again means the instance has the special attributes of the one subobject and of the another object. Here we have a contradiction because such a specification of a subobject does not exist or, if it would, the instance in question would belong to it and not to the other two subobjects.

The real intention to demand an ‘or’ relation is to allow the use of some background knowledge, which says that the two different formal specialization are semantically equivalent. An example: If a quadrangle specification is specialized to rectangles and rhombs then there might be an instance representing semantically a square. However, the use of the background knowledge heavily contradicts to the rules of using specifications. Concerning the example if the quadrangle is a matter of regard the it must be introduced as specification, for instance as a subtype of the rectangle and the rhomb. Anyway, the actual problem looking for an escape from the specification is an unsatisfactory specification. The attempt to establish an ‘or’ relation looks like.

Another attempt to interpret subobjects otherwise than as an ‘exclusive or’ relation is the introduction of an ‘and’ relation between subobjects. This means an instance of one subobject can only exist if there exists an instance of the another object. This condition looks like a global rule covering at least two subobjects. Evidently, there is an additional restriction between objects not defined by the specialization operation (inheritance). This restriction should better be defined as that what it is.

5.0 State and Failure Representation

On the contrary to the chapters before in this one there is not dealt with another class of mapping problems but with methods to represent information about the state of instances. This representation itself can cause ambiguities.

We present in the subsequent sections 5.2 at first a concept to represent the state of a single instance and then the extension to a set of instances or to global rules, respectively. We follow basically a concept worked out by Heinz During ([....]).

5.1 Classical Methods

The notification of failures used to be and is still a very individual matter. Often only with an additional explanation of the introduced principle another user than the author is able to interpret them. By far the failure representation is not specified formally as the description of the actual information is.

Two typical methods or combinations of them widely used with the test of physical instances. They are the following ones:

- There are introduced error codes indicating the overall result of the execution and giving more or less information about the kind of failures if there are ones.
- The physical instances are equipped with additional attributes with respect to the specification to carry information about failures.

In both cases there is no formal representation of the failure description, neither by applying the used specification language nor by introducing an own form. Regarding the problem of exchanging information via neutral models there is no approach seen to transfer the already available failure information about the physical instances as well.

It is necessary to develop a completely new approach for the representation of failures which should not be restricted to the description of failures but include the general representation of information about the instances. This information should be represented like the instances themselves so that the same principles for the transfer are applicable.

5.2 A State Information Concept

An essential goal of the concept developed consist in handling failure or state information, respectively, like the information which are the actual subject. That means, they are to specify, to instantiate and physically to manage like the basis information. Achieving this the state information can also be transferred via neutral models. Basically, the approach should be convenient, too, to provide failure information about the state information, but we will not go into this very theoretical subject.

The background why dealing with such a concept is that with various application areas it is preferred to transfer defective information together with the information about failures rather than to correct them before transferring.

5.2.1 The State Representation of Single Instances

The concept bases on a fundamental idea of object oriented languages: the multiple inheritance. It is a matter of how the apply this feature. The advantage of our approach with respect to neutral models comes into force if it is included into the specification of the neutral models. Otherwise the main problem is not met.

What we are interested in the state of a physical instance is how it corresponds to the valid specification, that means, to its object definition. This concerns as well violations of the specifications as any other characterizing information. Each of the attributes of an instance can be generally included in the state information. But it is even possible only to regard some of the attributes. It depends on the specification which is semantically desired.

This first part of the concept is restricted to state information which is observable and interpretable by testing one instance with respect to its specification. This includes potentially all the attributes and the local rules.

We introduce a general, separated structure in order to hold all the state information. This structure shall dynamically contain only those state information components which are relevant to the corresponding instance.

To demonstrate the concept we use again the EXPRESS language. But the concept does not depend on the language. It could be defined by using another object oriented language as well. The following extension of an arbitrary entity[1] is needed to realize the above outlined goal:

ENTITY an_arbitrary_object;

  state_information_component: SET [0:?][2] OF state_information;

 .....   (* the actual definition of the entity*);

END_ENTITY;

The used type state_information is to define as needed for the entity it is assigned to. It is to expect that sometimes it can be used repeatedly the same structure and sometimes it is needed possibly tailored for an entity type.

By using the inheritance concept it can be avoided to define the state_information_component attribute for each entity repeatedly if no special state representation is needed. It is to define once an entity of the following form

ENTITY general_state_specification;

  state_information_component: SET [0:?] OF state_information;

END_ENTITY;

1. We remind the term entity stands in EXPRESS for that what we call an object.
2. A set of between null and an undetermined number of elements is meant.

and to establish as a superentity of all the other declared entities. In the case that there is already one superentity of all the entities in question this one has only to be declared as a subentity of the general_state_specification entity. On this way on the top level of the entity specification a very general entity type for state_information can be introduced. The more the entities are specialized the more the type of state_information can be adapted. This concept has four major advantages:

- There is a specification of the state information available and the data created according to this specification follow the same principles like the actual data.
- The specification can be adapted to each structure caused by the semantics of the specification.
- The adaptation is only necessary in such an amount as it is really needed.
- The basic concept of the specification of the state information does not depend on the semantics of the actual information.

We give an example to illustrate the use of the concept.

Example 11: State Information for Point Entities

We use as the starting point parts of the point definition as realized in STEP and change and extend it to the need of our illustration goal. It is essential to know that there is an entity geometry standing as the root entity for all entities of the geometry type. That means the entity geometry is a superentity of the other geometrical entities.

We introduce at first a definition of the entity state_information which is assumed to taken over by the subentities of the point type.

```
ENTITY state_information;
  related_to: SET [1:?] OF attribute_identifyer;
  state_code: SET [1:?] OF state_description;
END_ENTITY;
```

The types attribute_identifyer and state_description are not detailed completely but could be imagined as any useful identification of an attribute and as an identification of possible events, respectively. The defined state_information type is as may be seen a splitting at first in the localization which attributes are concerned and at second what is the matter with it or them.

Now we define the top most entity geometry as a subentity of the entity general_state_specification. The latter one is used as defined above this example.

```
ENTITY geometry;
SUBTYPE OF (general_state_specification);
END_ENTITY;
```

The now following entities point and cartesian point inherit the state specification as defined.

```
ENTITY point;
SUBTYPE OF (geometry);
END_ENTITY;


ENTITY cartesian_point;
SUBTYPE OF (point);
  x_coordinate: REAL;
  y_coordinate: REAL;
  z_coordinate: REAL;
END_ENTITY;
```

So far the principle when inheriting the state specification from the general_state_specification type. If we would be interested to define a specific point state definition for any reason we can do it by introducing new types. Let us assume we want to characterize as a state information more precisely the points according to their location in relation to the origin of the coordinate system. Then we have to redefine the inherited attribute state_information_component in the entity cartesian_point as follows, for instances:

```
CONSTANT
  eps: REAL:= 10.0**-7;
END_CONSTANT;[1]
TYPE distance = ENUMERATION OF (close, far);[2]

ENTITY cartesian_point;
  state_information_component: distance;
  x_coordinate: REAL;
  y_coordinate: REAL;
  z_coordinate: REAL;
WHERE
  WR1: (distance = close) AND (SQRT(x_coordinate**2 + y_coordinate**2 + z_coordinate**2) < eps)
           OR (distance = far) AND (SQRT(x_coordinate**2 + y_coordinate**2 + z_coordinate**2) => eps)
END_ENTITY;
```

Analogously to the example there can be defined state information structures indicating a failure of the instance, for instance type errors or violations of local rules.

5.2.2 The State Representation of Sets of Instances

As already pointed out in the paragraph 3.2 there are constraints which can not be assigned to one single instance. The same situation we have with the state representation especially when the fulfillment or violation, respectively, of global rules is to indicate.

The basic idea for the state representation of instance sets is the same as used for single instances. Additionally it is needed a concept to summarize all those instances which are related to by an state information.

We use the general_state_specification as introduced in the paragraph before for the top most entity of the whole entity structure. Additionally it is defined a subentity of it to serve as the top most entity for the state specification of scopes of entities. We call this entity general_scope_specification and define it as follows:

1. This is the EXPRESS typical form of introducing constants.

2. Types which are not intended to be instantiated are introduced by the TYPE Definition and not by an ENTITY Definition.

```
ENTITY general_scope_specification;
SUBENTITY OF (general_state_specification);
  instance_set: SET [0:?] OF general_generic;
END_ENTITY;
```

general_generic is a general type expression. It stands for an arbitrary type or for a collection of types. Especially when general_generic is replace by the type general_state_specification then the attribute instance_set encloses all the defined entity types. Because the entity general_scope_specification is a subtype of general_state_specification the attribute state_information_component is inherited and may be used as generally defined or can be redefined to adapt better to the concrete structure. The same principle holds as in the single instance state case.

Depending on the desired application purpose selected other entities can be defined also as subentities of general_scope_definition. Then with the instances of those entities there is available also the information about the state of the sets of instances. But it is even possible to define special state entity as subentities of general_scope_specification to serve as information carrier about the state of sets of instances.

Analogously to the single instance state representation there are several advantages of the concept. Additionally to the above mentioned points we have here:

- There is a scope definition with represents the semantic scope.
- Even the state specification of sets of instances follows the present specification and instantiation concept.

To demonstrate the concept we give again an example.

Example 12: Rule for points

We use as far as possible the notation introduced in the example 11, especially the definition of point, cartesian_point (second definition), eps. We assume a rule is defined for cartesian points which expresses that exactly one instance has to exist lying within the sphere with the radius eps and with the center point in the origin. This rule may be defined formally as follows.

```
RULE point_match FOR (cartesian_point);
WHERE
  QERY (temp <*cartesian_point / temp.state_information_component = close) = 1;
END_RULE;
```

Now we can define a new entity which instance shall carry the information about the fulfillment or violation of the rule. We used already in the entity cartesian_point a redefined attribute state_information_component of the entity general_state_specification. In the following entity we introduce another redefinition of the same attribute and a redefinition of the attribute instance_set of the entity general_scope_specification.

```
ENTITY point_match_indicator;
SUBENTITY OF (general_scope_specification);
  state_information_component: INTEGER;
  instance_set: SET[0:?] OF (cartesian_point);
```

END_ENTITY;

This entity does not only specify the number of the closely to the origin located points but also those points which do it. In a case of a violation of the rule (more than one point) the violators are known.

Of course, a method is needed which determines the attribute of the entity point_match_indicator. The rule point_match alone is only the logical description of the constraint. But having specified the entity point_-match_indicator the implementation of any method is restricted to this specification. This is - demonstrated with an example - the advantage of this concept.

The examples exhibits the wide range of implementational freedom when realizing rules. This is true also with other kinds of state information. Therefore the example might be convenient, too, to illustrate the usefulness of having unambiguous state specifications.

6.0 Acknowledgment

The results presented in this paper are based essentially on very fruitful discussions we had mainly in 1991 with my former fellow-worker Felix Jacob Metzger (now with ETH Zurich, Switzerland) and with Heinz During (now with TU Berlin, Germany). They detected a lot of ambiguities while doing concepts and implementations of tests.

I would like to express my gratitude to the International Computer Science Institute for the excellent working conditions.

7.0 References

[1] Bertrand Meyer. Object-oriented Software Construction. Prentice Hall New York, London, Toronto, Sydney, Tokyo, 1988.

[2] Suad Alagic. Object-oriented database programming. Springer-Verlag, New York, Berlin, Heidelberg, London, Paris, Tokyo, 1989.

[3] Bjarne Stroustrup. The C++ Programming Language, Second Edition. Addison-Wesley Publishing Company, 1991.

[4] Bruce Eckel. Using C++, Covers C++ Version 2.0. Osborne McGraw-Hill, 1989.

[5] Stephen M. Omohundro. The Sather Language. Technical report, International Computer Science Institute, Berkeley, Ca., 1991.

[6] STEP (ISO 10303) part 11: EXPRESS Language Reference Manual. ISO TC 184/ SC4/ WG5 Document N14, 1991.

[7] STEP (ISO 10303) part 42: Integrated Generic Resources: Geometric & Topological Representation. ISO TC 184/SC4, Document N87, 1991.

[8] Heinz During. Konzeption und experimentelle Erprobung eines Verifikations-bausteines fuer STEP-Prozessoren [Concept and experimental realization of a verification component for STEP processors]. Diploma Thesis, Techn. Univ. Magdeburg, 1992

[9] Felix J. Metzger. Requirements for a graph of CAD instances. 2nd Intern. Conference on Computer Aided Design and Computer Graphics, Hangzhou, China, 1991, pp. 342-346.

[10] Felix J. Metzger. Verification: Semantic integrity of IPIM data. Working paper, Institute of Informatics and Computing Technique, Berlin, 1991.

[11] Dieter Richter. Concepts and problems concerning the implementation of STEP. 2nd Intern. Conference on Computer Aided Design and Computer Graphics, Hangzhou, China,1991, pp 334-341.

[12] Dieter Richter. Verification of STEP instances. Working paper, Physikalisch-Technische Bundesanstalt Berlin, 1991.

[13] Dieter Richter, Jurgen Heinrich. STEP-IPIM Speicher [STEP-IPIM repository]. IIR reports 7(1991), No.1