A Semantic Approach to Enforce Correctness of Data Distribution Schemes

J. A. Bakker

Department of Information Systems, Delft University of Technology, Julianalaan 132, 2628 BL Delft, The Netherlands

SQL, the Xplain data language and two entity-relationship approaches have been examined for their usability in the specification of data distribution. It is shown to be difficult to enforce correctness of fragmentation if all possible kinds of fragmentation are allowed. To solve this problem, a new approach to the design of data distribution schemes has been developed. This approach is based on a semantic model for fragmentation and allocation with additional restrictions. Only two kinds of horizontal fragmentation are allowed. Vertical fragmentation is not supported, though the possibility of local vertical fragmentation is not excluded.

Received July 1993; revised June 1994

1. INTRODUCTION

Many approaches to the specification of data distribution (fragmentation and allocation of data) are directed to an optimal solution for the distribution of data. They aim to minimize either response time or total costs or both. They have been reviewed, e.g. by Ceri and Pelagatti (1984), Apers (1988) and Özsu and Valduriez (1991). Knowledge about the use of data is also required in the top-down approach of Ceri et al. (1987). All approaches make it possible to design correct schemes for data distribution. Fragmentation is correct if disjoint and complete, allocation is correct if each fragment has at least one storage location (Özsu and Valduriez, 1991).

However, one problem has received too little attention: how to enforce correctness of data distribution schemes? This problem is associated with the large number of possible fragments, exceeding the number of relations in a data model. The complexity of this problem is illustrated by discussing some existing approaches to the specification of fragmentation and allocation.

To discuss existing approaches, an example of a data model is shown in Figure 1. It is partially related to the case described by Ceri and Pernici (1985). An abstraction hierarchy (Ter Bekke, 1992) is shown, which can be deduced from the informally presented relational model. In Figure 1, a line represents a semantic link (a reference). A line from the lower corner of a block of rectangles to the upper corner of a rectangle represents the IS-A relationship between disjoint specializations and their generalization (Smith and Smith, 1977a,b). ‘Production department’ and ‘sales department’ are disjoint specializations of ‘department’: a department may be a production or a sales department, but not both at the same time. Primary keys are underlined, foreign keys are shown in italic type face. It may be necessary to add restrictions to the model, e.g. if products should only be sold to customers residing in the region of a sales department, a restriction like ‘department.region = sale.customer.region’ should be added.

The data model describes data used in a company, assembling and selling computer systems. The company consists of four production departments (dept# 2–5) and two sales departments (dept# 6–7) located in different regions of a country.

One department (dept# 1) is the management department. Each department has a node and communicates with other nodes through a wide area network (Tanenbaum, 1988). Products are assembled by production departments and the composition of each product is registered. Basic materials are purchased from external suppliers (‘material order’). Supplies are recorded in relationship with the preceding material order. Production departments receive product orders from sales departments. The shipment of ordered products is recorded; planned and realized production quantities (‘production’) are registered, sales to customers are also recorded, and finally data on employees are recorded per department.

Additional restrictions are required to enforce disjoint specialization: a department may not be both a production department and a sales department. The next sections discuss existing approaches to the specification of fragmentation: SQL (Section 1.1), the Xplain language (Ter Bekke, 1991, 1992) (Section 1.2) and two approaches based on the entity-relationship (E-R) model (Chen, 1976) (Section 1.3). In Section 1.4 it is proposed to reduce the complexity of fragmentation schemes and in Section 1.5 the need for a meta level approach is discussed.

1.1. Fragmentation using SQL

It is assumed that a statement ‘FRAGMENT <name of fragment> IS’ is added to SQL. Some examples demonstrate that fragmentation can be related to the structure of an organization, i.e. the relation ‘depart-
ament' represents an organizational unit. The reason that almost all examples of fragmentation are simple is explained in Section 1.4.

1.1.1. No-fragmentation

All tuples of a relation can be grouped into one unconditional fragment. This no-fragmentation is a necessary preparation to allocation. An example of no-fragmentation:

FRAGMENT all-suppliers IS
supplier

A normalized relational meta model for relational concepts would make this statement superfluous: 'supplier' would be a tuple within the meta relation 'relation'. However, attempts to design a normalized relational meta model have not been successful (Ter Bekke, 1992). The relation 'supplier' also represents an external organizational unit.

1.1.2. Tuple fragmentation

A tuple fragment is a horizontal fragment based on the value of a primary key, e.g.:

FRAGMENT department-2 IS
department
WHERE dept# = 2

This is the smallest fragment possible if vertical fragmentation is excluded. The example also shows that tuple fragmentation can be related to the structure of an organization.

1.1.3. Conditional fragmentation

Conditional fragmentation is another kind of horizontal fragmentation, based on the value of an (inherited) attribute. An attribute value can be inherited through a semantic link built by a correct sequence of foreign and primary key pairs, e.g.

FRAGMENT products-of-department-5 IS
product
WHERE pdept# IN SELECT pdept# FROM department
WHERE dept# = 5.

Conditional fragmentation can be related to...
structure of an organization because the selected tuples of 'product' are connected to a tuple representing department 5. However, it is also possible to specify fragments based on irrelevant join operations.

1.1.4. Fragmentation based on irrelevant join operations

The following example of a connection trap illustrates the irrelevancy of a join for fragmentation:

\[
\text{FRAGMENT products-containing-material-of-supplier-71 IS} \\
p\# \text{ IN SELECT p\#} \\
\text{FROM component} \\
\text{WHERE m\# IN SELECT m\#} \\
\text{FROM material} \\
\text{WHERE} \\
sup\# = 71
\]

For each supplier a similar fragment can be specified. The shown connection trap (Codd, 1970; Date, 1990) contains one join condition in an opposite direction to the reference from a foreign key to the related primary key. However, there is no semantic link through which the attribute value 'material.sup\# = 71' is inherited by tuples of 'product'. For \( N \) tuples of 'supplier' and each product containing materials from various suppliers, each tuple of 'product' belongs to more than one of the \( N \) fragments. This means that disjointness of fragmentation cannot be guaranteed if conditional fragmentation is not based on a semantic link. It is even possible to use non-key attributes in a join operation, e.g.

\[
\text{FRAGMENT quickly-delivered-orders IS} \\
\text{shipment} \\
\text{WHERE date IN SELECT date} \\
\text{FROM product order}
\]

A product order can be pursued by several shipments. However, some values of 'shipment.date' can be missing among the values of 'product.order.date'. So, some tuples of 'shipment' would not belong to any fragment; completeness of fragmentation cannot be guaranteed if conditional fragmentation is not based on a semantic link. Other problems are associated with the use of operators other than '=' and the use of composite predicates.

1.1.5. Fragmentation based on a predicate not containing the '=' operator

If only two nodes are available in the company, the following fragment would make sense:

\[
\text{FRAGMENT employees-of-departments-1-to-4 IS} \\
\text{employee WHERE dept\# < 5}
\]

However, some problems are associated with this kind of fragmentation:

- Operators other than '=' make it difficult to guarantee disjoint and complete fragmentation.
- An increased number of nodes requires reallocation of tuples of 'employee', so fragmentation must be modified firstly. Restriction to the '=' operator avoids this last modification.

1.1.6. Fragmentation based on a composite predicate

An example of fragmentation based on a composite predicate:

\[
\text{FRAGMENT sales-of-sdept-7-of-products-of-pdept-4 IS} \\
sale \\
\text{WHERE sdept\# = 7} \\
\text{AND p\# IN SELECT p\#} \\
\text{FROM product} \\
\text{WHERE pdept\# = 4}
\]

Fragmentation based on composite conditions is difficult to check for disjointness and completeness. This checking becomes even more difficult if predicates contain operators other than '='. The number of irrelevant or undesirable fragments in SQL is very large and will interfere with the correctness of complex fragmentation schemes. A partial solution can be found in 'connection-trap-free' models (Chan, 1992).

1.2. Fragmentation using the Xplain language

Before discussing the Xplain language, the corresponding definition of the data model in Figure 1 is shown:

\[
type \text{department} = \text{department name, address, town, phone, region} \\
type \text{production department} = \text{[department], energy costs} \\
type \text{sales department} = \text{[department], advertising costs} \\
type \text{supplier} = \text{name, address, phone, town, country} \\
type \text{material} = \text{description, supplier, size, weight, price} \\
type \text{employee} = \text{name, initials, address, town, function, phone, salary, department} \\
type \text{product} = \text{product name, description, size, weight, price, stored_quantity, production department} \\
type \text{component} = \text{material, product, quantity} \\
type \text{production} = \text{product, first_date, last_date, planned_quantity, realized_quantity} \\
type \text{material order} = \text{material, production department, date, ordered_quantity} \\
type \text{supply} = \text{material order, date, quantity} \\
type \text{product order} = \text{product, ordered_quantity, date, sales department} \\
type \text{shipping} = \text{product order, date, quantity} \\
type \text{customer} = \text{name, address, town, region}
\]
type sale = product, sales department, customer, date, quantity, amount, delivery_date, status

Composite object types are placed left from '='. Square brackets indicate generalization. In Xplain a data model has an inherently defined structure. For example, 'sale its sales department' is an attribute of the composite type 'sale' and refers to the type 'sales department'. Static restrictions are specified by a derivable variable or an attribute from which the allowable value also can be specified. For example, the following assertion makes it possible to derive whether or not instances of the composite type 'sale' are correct:

assert sale its correct (true) =
(sales department its department its region = customer its region)

A statement 'fragment < name of fragment > is' can be added to Xplain for the specification of fragmentation as explained below.

1.2.1. No-fragmentation

For example, all instances of 'supplier' can be grouped into one fragment:

fragment all-suppliers is supplier

This definition can be regarded as superfluous because the concept 'type' is already defined by a meta model shown in Section 2: 'supplier' is an instance of the meta concept 'type'.

1.2.2. Instance fragmentation

An example of instance fragmentation:

fragment department-5 is department '5'

Here, 'department '5'' denotes the instance of 'department' with identification '5'. In this case the '=' operator is implicit.

1.2.3. Conditional fragmentation

An example of conditional fragmentation:

fragment production-of-department-5 is production
where product its production department its department = 5

1.2.4. The connection trap can be avoided

The specification of a semantic link requires to use the 'its' construct. For example: 'production where product its production department = 3'. The attributes in the semantic link used here are 'production its product' and 'product its production department', both containing the type 'product'. This required overlap between attributes makes incorrect sequences of attributes detectable. The correctness of a link is easily determined because a link must contain valid attributes, e.g. 'product its production' is not an attribute in our data model.

Xplain avoids connection traps, but allows the use of composite predicates. Therefore, the number of syntactically correct fragments is still too large for disjointness and completeness checks.

1.3. Some E-R approaches

The approach of Bertino (1983) extends the concepts of the E-R model (Chen, 1976). Here, both fragments and entity-types are specified in an extended data model. Therefore, insertion or deletion of, for example, a fragment like 'department 9' requires modification of this extended data model. Further, correctness of schemes cannot be considered in this approach. Due to the level of specification, an invariant description of concepts for fragmentation, not susceptible to changes in the database, cannot be developed in this way.

A generic E-R approach by Ceri et al. (1989) is based on a meta model containing concepts for data modelling (e.g. 'global entity' and 'global attribute'), data distribution, monitoring the use of fragments and communication links. An entity 'fragmentation operation' represents algebraic operations and is connected to itself via a relationship 'arc'. Composite predicates are allowed by this meta model, therefore containing almost all foregoing problems mentioned for SQL. Furthermore, 'fragmentation operation' is not linked to the entity 'global attribute'. This allows a non-conditional fragment like 'SELECT * FROM material'. However, this last specification technique is redundant because its result is equivalent to 'material which can also be regarded as an instantiation (tuple) of 'global entity'. Therefore, this E-R meta approach is not consistent. Furthermore, it does not consider the enforcement of correctness of data distribution schemes.

1.4. Reduction of the complexity of fragmentation schemes

Fragmentation must be disjoint and complete, and each fragment must have at least one location (Özsu and Valduriez, 1991). The criterion of reconstruction is not relevant here as global vertical fragmentation will be excluded to reduce the complexity of fragmentation schemes. It is assumed that the number of possible kinds of fragmentation should be reduced before enforcement of correctness of data distribution schemes can be endorsed. The following two assumptions are made to reduce the complexity of fragmentation schemes.

1.4.1. Global vertical fragmentation is not allowed

If the number of non-primary key attributes in a relation (n) is very large, the number of possible attribute groups (N) approximates \( n! \) (Hammer and Niamir, 1979). If \( n = 10 \) then \( N \) approximates 115,000 (Navathe et al., 1984). In complex data models, the number of possible vertical fragments to be evaluated for an optimal grouping, is therefore too large. This problem can be avoided by
complete local attribute partitioning. Tuples of a relation having $Q$ attributes, can be stored in $Q$ transposed or attribute files (Batory, 1979; Teorey and Fry, 1982). Replication of key values can be avoided by using the same logical position for all attribute values of a tuple in different files. In that case, tuples can be found via one index per relation. The use of transposed files in the centralized Xplian DBMS (Ter Bekke et al., 1991), combined with B*-trees (Bayer et al., 1972, 1977; Comer, 1979), shows good performance in general. However, insertion or retrieval of a tuple takes at least $Q$ disk accesses. To reduce the time for disk access, disk arrays can be used for the local distribution of attribute files (Weikum et al., 1992). A reduced complexity of distribution schemes and a reduced number of local access structures are arguments for local attribute partitioning and against global vertical fragmentation. This advantage increases with the number of nodes and the number of attributes: to enhance local autonomy for data processing, each node must have a copy of the data distribution schemes. Transposed files often reduce the number of disk accesses because many queries search only a small number of attributes, or conditions apply only to one attribute.

1.4.2. The number of kinds of fragmentation must be further limited

Fragmentation can be based on semantic links because these links represent in many cases the structure of an organization. For example, in Figure 1 contains many semantic links to 'sales department' and to 'production department'. Fragmentation can therefore be based on (nested) single conditions containing the '=' operator.

In a similar way, clustering of data in centralized database systems can be based on semantic links (Chang and Katz, 1989; Bakker, 1992; Cheng and Hurson, 1992). The use of more than one semantic link in conditional fragmentation makes it difficult to check for correctness of fragmentation. To reduce the complexity of fragmentation schemes it is proposed that only the following three kinds of fragmentation, partially based on (the implicit) '=' operator, are allowed:

- Tuple fragmentation (instance fragmentation).
- Conditional fragmentation based on a semantic link.
- No-fragmentation (type allocation).

1.5. Further discussion of existing approaches

It still remains necessary, within the limitations chosen in Section 1.4, to discuss whether or not existing approaches offer enough support to enforce correctness of data distribution schemes. This correctness should be based on the well known criteria: disjointness and completeness of fragmentation, and minimal allocation of fragments.

1.5.1. Disjointness of fragmentation

The following fragments are not disjoint: 'FRAGMENT departments IS department' and 'FRAGMENT department-3 IS department WHERE dept1 = 3'. In case of a specific database system, software can detect such non-disjointness of fragments. However, it is not easy to design generic software able to check disjointness irrespective of the data model. Another problem is the correctness of specified links. A 'flat' join, containing many join predicates is more difficult to check than a nested 'IN' or 'EXISTS' construct, also possible in SQL. Further, a semantic link should not be based only on equal domains of involved keys, but a correct match of these keys is also required. For each relation or composite type the fragmentation should be registered to enforce disjointness within the chosen fragmentation. A generic approach needs formal concepts (meta meta data), i.e. the concept 'relation' or 'type' must be extended with a property 'kind of fragmentation'. Absence of such concepts implies that disjointness of fragmentation can only be addressed at the level of the meta data (data model) of a specific distributed database system.

1.5.2. Completeness of fragmentation

If a tuple is inserted, how can a D-DBMS determine whether a tuple fragment and its location has to be specified first? In case of tuple fragmentation the number of tuple fragments must equal the cardinality of the involved relation. In case of conditional fragmentation (e.g. grouping of employees per department) the number of fragments must equal the cardinality of the referred relation. For example, $N$ tuples of 'department' require $N$ groups of 'employee' to be specified. However, data manipulation languages have not been designed for the purpose of counting fragments. These problems are probably solvable for a specific distributed database. A generic approach should be based on an invariant description of allowable kinds of fragmentation and restrictions imposed on the number of fragments.

1.5.3. Minimal allocation of fragments

The discussed languages and the approach of Bertino (1983) do not support concepts like 'fragment location'. We need new concepts to be able to count the number of locations per fragment and to check that this number is at least one.

1.5.4. Preliminary conclusion

Data manipulation languages can only address the meta data of a data model and the predicates possibly associated with fragmentation. Because a normalized relational meta model does not exist, a relational meta model solution with new concepts for the specification of data distribution is impossible. Even for the design of a specific distributed database system, relational approaches offer only a partial support for the enforcement of correctness of data distribution schemes. The present paper, the paper and book of Ter Bekke (1980,
2. CONCEPTS FOR DATA MODELLING

It is necessary to discuss shortly the semantic concept for data modelling because they are fundamental to the new concepts for the specification of fragmentation. Concepts for data modelling are derived from Xplain (Ter Bekke, 1992). The definition of the composite concepts 'type' and 'attribute' in Figure 2 consists of a unique name followed by '=' and a unique list of properties. An abstraction hierarchy is also shown here. Its structure is derivable from the type definitions. Though 'type' can be regarded as a composite concept, it also occurs in two properties of the concept 'attribute': 'attribute is composite type' and 'attribute its type'. The attribute 'type its name' can be changed without changing the identification of a type and related referring attribute values. It is supposed that object identifications may not be modified (Khoshafian and Copeland, 1986).

An explanation of these meta concepts for data modelling is supported by the data model in Section 1.2. For example, 'name' and 'product' are instances of the concept 'type'. Types only mentioned as attributes, e.g. 'date', are base types.

An attribute is specified by the name of its type and may have a prefix indicating a role. If attributes of a composite type have the same type, they must be distinguished by roles, e.g. 'production its first_date' and 'production its last_date'. The attribute 'attribute is kind' makes it possible to distinguish aggregation and specialization as defined by Smith and Smith (1977). The concepts of Xplain result in an inherent relatability (referential integrity) of data models (Ter Bekke, 1980, 1992). For example, 'department' occurs as a composite type and as the attribute 'employee its department'. Xplain also supports additional static restrictions, based on controlled redundancy. An example, using the derivable attribute 'department its number of managers', states that each department should have exactly one manager:

\[
\text{assert department its number of managers (1..1) = count employee where function = 'manager' per department}
\]

Tables 1 and 2 contain some instances of 'type' and 'attribute'. The structure of these Tables is derived from the meta model shown in Figure 2. The left columns of

<table>
<thead>
<tr>
<th>Table 1. Examples of types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>21</td>
</tr>
<tr>
<td>22</td>
</tr>
<tr>
<td>23</td>
</tr>
<tr>
<td>24</td>
</tr>
<tr>
<td>25</td>
</tr>
<tr>
<td>29</td>
</tr>
</tbody>
</table>

1992) show pitfalls and weaknesses of relational approaches. This implies that the statement "ease of fragmentation and ease of reconstruction are two of the many reasons why distributed systems are relational" (Date, 1990) cannot be justified nowadays. A solution might be to modify relational concepts in such a way that a normalized meta model for relational concepts can be developed and to extend this meta model with new concepts for the specification of data distribution. Here we choose to add new concepts for the specification of fragmentation and allocation to a meta model for data modelling, derived from Xplain.

The following Sections summarize the concepts of Xplain (Section 2), and show new concepts for fragmentation (Section 3) and allocation (Section 4). Enforcement of correctness of data distribution schemes through restrictions is discussed in Section 5. The usability of these restrictions is discussed in Section 6. Conclusions and research questions are discussed in Section 7.

<table>
<thead>
<tr>
<th>Table 2. Examples of attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>31</td>
</tr>
</tbody>
</table>
both Table contain identifications of meta data belonging to the data model shown in Figure 1.

The value 'A' for 'attribute is kind' indicates aggregation and specialization is indicated by 'S' (Table 2). A short explanation is placed between curly brackets.

Most of the shown attributes will be chosen to participate in conditional fragmentation (Section 3.1). For example, a semantic link 'production its product its production department' ends with attribute '11': 'production its product', which is preceded by attribute '10': 'product its production department'. The next Section shows that the concepts for fragmentation are depending on these concepts for data modelling.

3. CONCEPTS FOR FRAGMENTATION

Three kinds of fragmentation are considered: conditional fragmentation (Section 3.1), instance fragmentation (Section 3.2) and no-fragmentation (Section 3.3).

3.1. Conditional fragmentation

Conditional fragmentation is based on the concepts 'path attribute', 'group' and 'instance', which are defined in Figure 3 (Bakker, 1992).

To enforce that for each composite type only one kind of fragmentation is specified, 'type its fragmentation' is added to the definition of 'type'. This addition has no relevance for data modelling. The kind of fragmentation must be specified before any path attribute or group can be defined. During data modelling, default values for 'type its fragmentation' can be used. Different default values for composite types (e.g. '+' and base types (e.g. '-') are necessary because fragmentation within base types is not required. Before specifying fragments, '+' values must be substituted by one of the three values:

- 'C': conditional fragmentation (this section)
- 'I': instance fragmentation (Section 3.2)
- 'N': no-fragmentation (Section 3.3)

The value 'C' requires that path attributes, referred instances and groups are specified. The members of a group inherit a property through a common path, e.g. some instances of 'production' have a common path 'production its product its production department = 4', containing 'production its product' as the second path attribute and 'product its production department' as the first path attribute. The referred instance is 'production department '4'.

The correctness of a path can be checked: 'path attribute its attribute its type' must be the same as 'path attribute its preceding path attribute its attribute its composite_type'. For example, in the path 'production its product its production department' the composite type of the first path attribute is 'product', which is also the type of the second path attribute. A first path attribute is not preceded by a path attribute. This is indicated by a value '0' for 'path attribute its preceding path attribute'. The definition of 'path attribute' is recursive because paths can have different lengths and cannot be specified only by the first and last attribute. For example, Figure 4 shows a data model containing two paths between 'activity' and 'department', both starting with 'employee its department' and ending with 'activity its participant'. Some path attributes are shown in Table 3.

The left column of Table 3 contains the arbitrary identification of some path attributes.

A path can be part of a longer path, e.g. path attribute '1' not only defines a path between 'product' and 'production department', but is also the first path

---

**FIGURE 3.** Meta model for conditional fragmentation.

**FIGURE 4.** An example of partially coinciding paths.
TABLE 3. Examples of path attributes

<table>
<thead>
<tr>
<th>Path attribute</th>
<th>Preceding path attribute</th>
<th>Attribute</th>
<th>Textual definition of 'path attribute its attribute'</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>10</td>
<td>product its production department</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>11</td>
<td>production its product</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>9</td>
<td>employee its department</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>31</td>
<td>customer its region</td>
</tr>
</tbody>
</table>

TABLE 4. Types having conditional fragmentation

<table>
<thead>
<tr>
<th>Type (its name)</th>
<th>Definition of an attribute path</th>
</tr>
</thead>
<tbody>
<tr>
<td>supply</td>
<td>supply its material order its production department</td>
</tr>
<tr>
<td>material order</td>
<td>material order its production department</td>
</tr>
<tr>
<td>component</td>
<td>component its product its production department</td>
</tr>
<tr>
<td>production</td>
<td>production its product its production department</td>
</tr>
<tr>
<td>product</td>
<td>product its production department</td>
</tr>
<tr>
<td>product order</td>
<td>product order its product its production department</td>
</tr>
<tr>
<td>sale</td>
<td>sale its sales department</td>
</tr>
<tr>
<td>customer</td>
<td>customer its region</td>
</tr>
<tr>
<td>employee</td>
<td>employee its department</td>
</tr>
<tr>
<td>shipment</td>
<td>shipment its product order its product its production department</td>
</tr>
</tbody>
</table>

TABLE 5. Examples of referred instances

<table>
<thead>
<tr>
<th>Instance</th>
<th>Type (its name)</th>
<th>Value</th>
<th>Textual definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>production department</td>
<td>3</td>
<td>production department '3'</td>
</tr>
<tr>
<td>6</td>
<td>production department</td>
<td>4</td>
<td>production department '4'</td>
</tr>
<tr>
<td>9</td>
<td>department</td>
<td>4</td>
<td>department '4'</td>
</tr>
<tr>
<td>10</td>
<td>department</td>
<td>5</td>
<td>department '5'</td>
</tr>
<tr>
<td>13</td>
<td>region</td>
<td>west</td>
<td>region 'west'</td>
</tr>
<tr>
<td>14</td>
<td>region</td>
<td>east</td>
<td>region 'east'</td>
</tr>
</tbody>
</table>

TABLE 6. Examples of groups

<table>
<thead>
<tr>
<th>Group</th>
<th>Last path attribute</th>
<th>Ref. instance</th>
<th>Textual definition of the predicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>9</td>
<td>employee its department = 4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>10</td>
<td>employee its department = 5</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>5</td>
<td>product its production department = 3</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>6</td>
<td>product its production department = 4</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>5</td>
<td>production its product its production department = 4</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>6</td>
<td>customer its region = 'east'</td>
</tr>
<tr>
<td>25</td>
<td>6</td>
<td>14</td>
<td>customer its region = 'east'</td>
</tr>
<tr>
<td>26</td>
<td>6</td>
<td>13</td>
<td>customer its region = 'west'</td>
</tr>
</tbody>
</table>

attribute of a path 'production its product its production department', consisting of the path attributes '1' and '2'. Conditional fragmentation is proposed for the composite types shown in Table 4. These types have a semantic link to types like 'production department' and 'region' which are related to units defined by the example organization. Table 5 shows some referred instances.

Table 6 shows that different conditional fragments may exist within the same composite type. For example: group '1' and group '2' are fragments of the composite type 'employee' both based on the path 'employee its department' but related to different referred instances of 'department'. The referred instances have the identifications '9' and '10' (Table 5).

Additional restrictions to enforce correctness of fragmentation are discussed in Section 5.3.

3.2. Instance fragmentation

Instances of 'department' are not frequently used for daily operational activities. Therefore, it seems not to be very important which kind of fragmentation and allocation is chosen. However, considering management of semantic integrity, it makes sense to locate an instance of 'department' on a node where local data refers to this instance.

In this way, data processing for the maintenance of integrity can be kept local. The same argument applies to
choosing instance fragmentation for 'sales department' and 'production department'. After specifying 'T' for 'type its fragmentation', it is necessary to specify the instances of the types having instance fragmentation. The already introduced concept 'instance' can be used: type instance = type, value. Additional restrictions are discussed in Section 5.1, e.g. the number of instances of 'instance' must equal the cardinality of the involved type with instance fragmentation.

3.3. No-fragmentation

As there is no semantic link in our data model between 'supplier' and any type representing an organizational unit, fragmentation of 'supplier' cannot be based on such a link. It is assumed that a department is not obligated to purchase materials in its town, so fragmentation cannot be based on 'supplier its town'. Instance fragmentation also makes no sense because of the very large number of instances of 'instance' to be specified. To avoid this overhead, a choice is to apply no-fragmentation for 'supplier'. It makes sense that all instances of 'supplier' are allocated to the nodes of the production departments. This is an example of replication (Section 4).

A remaining problem is the fragmentation of 'material'. This type has no semantic link to any organizational unit like 'production department'. Further it is assumed that each supplier can supply many materials. So, the cardinality of 'material' is higher than that of 'supplier'. A choice is to apply no-fragmentation ('N') and to allocate all instances of 'material' to the nodes of production departments. Though the overhead of instance fragmentation is avoided, a disadvantage is that some instances of 'material' are replicated at departments not using the corresponding materials. Examples of no-fragmentation are shown in Section 4, and additional restrictions are discussed in Section 5.2.

4. CONCEPTS FOR ALLOCATION

Concepts for allocation, 'instance location', 'group location' and 'type location', are shown in the meta model of Figure 5, which also supports many locations per fragment (replication).

The definition of the meta model of Figure 5 is:

<table>
<thead>
<tr>
<th>Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>type node</td>
<td>address, location</td>
</tr>
<tr>
<td>type type</td>
<td>name, domain, fragmentation</td>
</tr>
<tr>
<td>type attribute</td>
<td>composite_type, type, role, kind</td>
</tr>
<tr>
<td>type instance</td>
<td>type, value</td>
</tr>
<tr>
<td>type instance</td>
<td>instance, node</td>
</tr>
<tr>
<td>type path</td>
<td>preceding_path attribute, attribute</td>
</tr>
<tr>
<td>type group</td>
<td>last_path attribute, ref_instance</td>
</tr>
<tr>
<td>type type</td>
<td>composite_type, node</td>
</tr>
<tr>
<td>type location</td>
<td>group, node</td>
</tr>
</tbody>
</table>

The value of 'node its address' identifies a node. So, an address can be changed without changing the identification of an instance of 'node' and the references to this

<table>
<thead>
<tr>
<th>Node</th>
<th>Address</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>34561246</td>
<td>at department 1</td>
</tr>
<tr>
<td>7</td>
<td>34560010</td>
<td>at department 7</td>
</tr>
</tbody>
</table>
instance of 'node'. The value of 'group its ref_instance its value' must be known before any instance of 'group' or 'instance' can be inserted. Because groups are based preferably on (inherited) attribute values representing organizational units, this value can be known beforehand. The same holds for instance fragmentation. Tables 7–11 show applications of the meta model of Figure 5. To keep the examples understandable, the identification of a node equals the identification of a department.

Table 9 shows some instances of 'group location'. To support understandability, the identification of a production or a sales department equals the identification of the belonging department. The referred groups can be found in Table 6 (Section 3.1).

Table 10 shows some instances of 'instance' belonging to composite types. The shown instances are designed as fragments, but they can also be referred to by instances of 'group' through 'group its ref_instance'. Now 'instance its value' corresponds with the identification of instances of composite types. Some instances of base types have been shown in Table 5. Some instances of 'instance location' are shown in Table 11.

Some types (instances of 'type') including the kind of fragmentation are shown in Table 12.

The former sections demonstrate that meaningful fragmentation and allocation can be supported by the proposed meta model. However, the meta model cannot enforce that fragmentation is disjoint and complete. Furthermore, a minimal allocation of fragments cannot be enforced by the meta model. The next section deals with additional static restrictions required for the enforcement of correctness of fragmentation and allocation.

### Table 8. Examples of the allocation (replication) of a composite type

<table>
<thead>
<tr>
<th>Location</th>
<th>Composite type</th>
<th>Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>25 (supplier)</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>25 (supplier)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>All instances of 'supplier' at the node of department 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>All instances of 'supplier' at the node of department 4</td>
<td></td>
</tr>
</tbody>
</table>

### Table 9. Examples of the allocation of groups

<table>
<thead>
<tr>
<th>Location</th>
<th>Group</th>
<th>Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>33</td>
<td>25</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Data on employees of department 4 at node 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data on employees of department 5 at node 5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data on products of production department 4 at node 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data on production by production department 3 at node 3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data on customers of region 'east' at node 6</td>
<td></td>
</tr>
</tbody>
</table>

### Table 10. Some instances of composite types

<table>
<thead>
<tr>
<th>Instance</th>
<th>Type (its name)</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>21 (production department)</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>7 (department)</td>
<td>4</td>
</tr>
<tr>
<td>22</td>
<td>29 (sales department)</td>
<td>6</td>
</tr>
<tr>
<td>16</td>
<td>7 (department)</td>
<td>6</td>
</tr>
</tbody>
</table>

### 5. Enforcement of Correctness of Data Distribution Schemes

The enforcement of correctness can be based partially on the meta model. The principle of convertibility in Xplain (Ter Bekke, 1980) implies that the collection of attribute values belonging to an instance of a composite type identifies this instance as well as the single identification of this instance.

In this way, redundant specification of instances of composite types can be avoided. Of course, the meta model requires correctness of references. However, the meta model permits specifications conflicting with the correctness of data distribution schemes, e.g.

- A path attribute referring to a non-preceding path attribute.
- A type location referring to a type having conditional or instance fragmentation.
- An instance location referring to a type not having instance fragmentation.

Furthermore, the meta model cannot enforce that for all fragments (instances of 'instance', 'type' or 'group') at least one location is specified. A further discussion of this subject will be presented in the following three sections. The first static restrictions to be dealt with are concerned with the correct specification of 'type its fragmentation'. The virtual attribute 'type its composite' must be derived before it can be determined whether or not the kind of fragmentation of types has been defined correctly.

1. assert type its composite = any attribute per composite_type

After defining a data model, all values '+' for the kind of fragmentation must be replaced by one of the three allowed values. For base types, 'type its fragmentation must be '+-' (assertion 2). The assertions 3–5 are required as a basis for the resuming assertions 12, 15 and 25:

2. assert correctness of base types (true) = nil type where not composite and not (fragmentation = '—')
## Table 11. Examples of instance locations

<table>
<thead>
<tr>
<th>Instance location</th>
<th>Instance</th>
<th>Node</th>
<th>Comment on the location of instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
<td>4</td>
<td>'production department' '4' at node 4</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>4</td>
<td>'department' '4' at node 4</td>
</tr>
<tr>
<td>8</td>
<td>22</td>
<td>6</td>
<td>'sales department' '6' at node 6</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>6</td>
<td>'department' '6' at node 6</td>
</tr>
</tbody>
</table>

## Table 12. Examples of types including fragmentation

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Domain</th>
<th>Fragmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>region</td>
<td>CHAR12</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>department</td>
<td>INT3</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>employee</td>
<td>INT5</td>
<td>C</td>
</tr>
<tr>
<td>13</td>
<td>date</td>
<td>INT6</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>production</td>
<td>INT3</td>
<td>I</td>
</tr>
<tr>
<td>22</td>
<td>product</td>
<td>INT3</td>
<td>C</td>
</tr>
<tr>
<td>23</td>
<td>production</td>
<td>INT6</td>
<td>C</td>
</tr>
<tr>
<td>24</td>
<td>customer</td>
<td>INT4</td>
<td>C</td>
</tr>
<tr>
<td>25</td>
<td>supplier</td>
<td>INT3</td>
<td>N</td>
</tr>
</tbody>
</table>

[3] assert type its instance fragmentation = composite and (fragmentation = 'Y')

[4] assert type its no-fragmentation = composite and (fragmentation = 'N')

[5] assert type its conditional fragmentation = composite and (fragmentation = 'C')

It is also required that a distributed database system contains at least one node:

[6] assert number of nodes (1..*) = count node

The following three sections deal with assertions necessary to enable a D-DBMS to check and enforce correctness of data distribution schemes. Each kind of fragmentation and allocation requires different assertions to enforce correctness of data distribution schemes.

### 5.1. Correctness of instance fragmentation and allocation

In case of instance fragmentation, an instance of 'instance' must refer to a composite type having instance fragmentation, 'instance its type its instance fragmentation' must be true.

[7] assert instance its correct reference = type its instance fragmentation

However, a value restriction for 'instance its correct reference' is not included here, because some instances of 'instance' belong to base types and exist only to be referred to by groups. For example, some groups refer to instances of the base type 'region', e.g. conditional fragmentation within 'customer' based on 'customer its region = 'west' '.

Further, each instance of 'instance location' must refer to an instance of 'instance' belonging to a composite type having instance fragmentation:

[8] assert instance location its correct reference (true) = instance its correct reference

The meta model requires that deletion of an instance of a composite type having instance fragmentation includes deletion of the related instance of 'instance'. This deletion must be preceded by the deletion of all related instances of 'instance location'. The following assertion is a preparation to enforce the specification of a minimal number of instance locations (see also assertion 12):

[9] assert instance its number of locations = count instance location per instance

A value restriction (1..*) is not included here because this does not apply to instances of base types; it is expressed in restriction 12. The number of instances of 'instance' referring to a certain composite type must equal the number of instances of this type: 'type its cardinality' (assertions 10 and 11). The value of this cardinality is derivable from the contents of a database. However, this derivation cannot be specified declaratively because not all instances of composite types can also be found as instances of 'instance'. Even so, 'type its cardinality' can be determined by a D-DBMS. The following two assertions are needed to check the number of instances of composite types:

[10] assert type its number of instances = count instance per type

[11] assert type its correct number of instances = (cardinality = number of instances)

Value restrictions are not specified here because they only apply to types with instance fragmentation. Correctness of instance fragmentation and allocation
can now be specified by the derivable variable 'correctness of instance fragmentation and allocation' (assertion 12). The meta model requires that each instance location has a node, so assertion 12 depends on assertion 6.

\{12\} assert correctness of instance fragmentation and allocation (true) =
\begin{align*}
& ((\text{nil type}) \\
& \quad \text{where instance fragmentation} \\
& \quad \text{and not correct number of instances}) \\
& \quad \text{and} \ \\
& (\text{nil instance}) \\
& \quad \text{where correct reference} \\
& \quad \text{and number of locations < 1}) \text{ and} \\
& (\text{nil instance location}) \\
& \quad \text{where not correct reference})
\end{align*}

For example, the insertion of the instance 'department '9' must be executed as a transaction starting with the insertion of a correct instance of 'instance' (instance its type = 7 (department), instance its value = 9). This instance of 'instance' can be derived by a D-DBMS.

Furthermore, at least one instance of 'instance location' correctly referring to the mentioned instance of 'instance' must be specified. This specification should be done by an expert user. Another example is the deletion of an instance of 'department'. Such a deletion, if not violating referential integrity, must be followed by a deletion of all related instances of 'instance location' and the related instance of 'instance'. This sequence is determined by the meta model. A D-DBMS can derive which instance of 'instance' and instances of 'instance location' are involved in this sequence of deletions.

5.2. Correctness of no-fragmentation (type allocation)

The meta model requires the following assertions to be evaluated in the presented order. First, an instance of 'type location' must refer correctly to a composite type:

\{13\} assert type location its correct reference (true) =
\begin{align*}
\text{composite_type its no-fragmentation}
\end{align*}

Each unfragmented composite type must have at least one instance of 'type location'. A value restriction is not specified here, because this can only apply to unfragmented composite types:

\{14\} assert type its number of type locations =
\begin{align*}
\text{count type location per composite_type}
\end{align*}

The derivable variable 'correctness of no-fragmentation and allocation' enables to check for correctness of no-fragmentation and allocation (assertion 15). The meta model implies that assertion 15 depends on assertion 6: each type location refers to a node.

\{15\} assert correctness of no-fragmentation and allocation (true) =
\begin{align*}
& (\text{nil type}) \\
& \quad \text{where no-fragmentation}
\end{align*}

and number of type locations < 1) and
(nil type location
\quad \text{where not correct reference})

For example, each unfragmented composite type, e.g. 'supplier', must have at least one associated instance of 'type location'. The removal of an unfragmented composite type must include the removal of all associated instances of 'type location', which must be preceded by the deletion of all instances of this composite type. The order of deletions can be derived from the meta model. In case of adding a composite type with no-fragmentation to an existing data model, the order of required insertions ('type', 'type location') can also be derived.

5.3. Correctness of conditional fragmentation and allocation

The meta model requires a certain order for inserting instances of the following meta data: 'path attribute', 'group' and 'group location'. First, some references must be specified correctly. Each instance of 'path attribute' must have a correct predecessor. The predecessor is correct if there is no predecessor or if the type of a path attribute equals the composite type of its predecessor:

\{16\} assert path attribute its correct predecessor (true) =
\begin{align*}
\text{preceding_path attribute = 0 or} \\
\text{attribute its type = preceding_path} \\
\text{attribute its composite_type}
\end{align*}

For example, in the path 'employee its department', there is no preceding path attribute. Grouping within 'production' is based on the path 'production its product its production department'; the path attribute 'production its product' has the predecessor 'product its production department'. These path attributes are connected through the type 'product'. It is also necessary that a group belongs to a composite type with conditional fragmentation. This type can be found through 'group its last_path attribute its attribute its composite_type'. Therefore, 'group its last_path attribute its attribute its composite_type its conditional fragmentation' must be true:

\{17\} assert group its correct fragmentation (true) =
\begin{align*}
\text{last_path attribute its attribute its composite_type its conditional fragmentation}
\end{align*}

The inherited type of a group is 'group its ref_instance its type'. For example, group '4' is based on the condition 'product its production department = 3' (Table 6). The inherited type of group '4' is 'production department', which is also the type of the referred instance 'production department 3'. The traversal of a path of which the number of path attributes depends on a data model is required to derive the value of 'group its inherited type':

\{18\} assert group its inherited type =
\begin{align*}
\text{last_path attribute its preceding_path}
\end{align*}
attribute its preceding_path attribute its
........................ preceding_path attribute its
attribute its type

However, the variable length of path's impairs a
declarative specification of this derivation. Nevertheless,
the value of 'group its inherited type' is derivable by a D-
DBMS. Both ways to determine the inherited type of a
group must have the same result:

(19) assert group its correct references (true) =
(inherited type = ref_instance its type) and
correct fragmentation

In addition to correctness of references required by the
meta model, the preceding assertions specify which
references have to be checked and which are correct.
Now it is necessary to discuss the enforcement of
completeness of specification of instances of 'path
attribute', 'group' and 'group location'. Assertions 20
and 21 are a preparation to assertion 25: the conditional
fragmentation of a composite type must be based on
exactly one predicate as defined by a path of attributes.
Assertion 20 derives whether a path attribute functions
as the last path attribute in a path. This derivation makes
it possible to determine the number of specified
predicates per composite type:

(20) assert path attribute its number of groups =
count group
per last_path attribute

(21) assert type its number of predicates =
count path attribute
where number of groups > 0
per attribute its composite_type

Assertion 21 does not contain a value restriction as
this does not apply to composite types not having
conditional fragmentation. However, such a restriction
must be present in the assertion, which addresses
correctness of conditional fragmentation and allocation
(25). For example, instances of 'product order'
are grouped on the basis of the path 'product order its
product its production department' containing 'product
order its product' as the last path attribute. A
simultaneous grouping within 'product order' based on
another path, e.g. 'product order its sales department',
must be excluded to avoid non-disjoint fragmentation.
It is also required that each group has at least one location:

(22) assert group its number of locations (1..*) =
count group location
per group

For example, if there are N instances of 'production
department', then N groups must exist within each of the
types 'product', 'production', 'material order', 'ship-
ment', 'product order', 'component' and 'supply'. The
same rule for the number of groups can also be expressed
through a derived property of 'path attribute' because all
groups within a composite type must be based on the
same last path attribute. This makes it possible to specify
the following assertion, using the cardinality of the
inherited type:

(23) assert path attribute its derived number of groups
= preceding_path attribute its preceding_path attribute its
........................ preceding_path attribute its
attribute its type its cardinality

These two ways (assertions 20 and 23) to count the
number of groups per path must have the same result.
This will be expressed in assertion 25. Before complete-
ness of instances referred to by groups can be checked, it
is necessary to determine whether a type is inherited:

(24) assert type its inherited =
any path attribute
where preceding_path attribute = 0
per attribute its type.

If 'type its inherited' is true, completeness of specifica-
tions of instances of 'instance' referred to by instances of
'group' must be checked (see assertion 25). Correctness
of conditional fragmentation and allocation can now be
expressed:

(25) assert correctness of conditional fragmentation
and allocation (true) =
((nil type
where conditional fragmentation
and not number of predicates = 1) and
(nil type
where inherited
and not correct number of instances)
and
(nil path attribute
where not correct predecessor
or not (number of groups = derived
number of groups)) and
(nil group
where number of locations < 1
or not correct references))

The meta model implies that assertion 25 depends
on assertion 6: each group location refers to a node.
The attribute 'type its correct number of instances'
has been derived in Section 5.1. It makes no sense to
require 'nil group location where not group its correct
references' because assertion 19 prevents the possibility
of specifying any incorrect group. Finally, it is possible
to express correctness of a data distribution scheme
through the derivable variable 'correctness of data
distribution':

(26) assert correctness of data distribution (true) =
(correctness of instance fragmentation
and allocation) and
(correctness of no-fragmentation and
allocation) and
(correctness of conditional fragmentation
and allocation)
6. DISCUSSION

The specification of assertions through derivable variables or attributes and the use of these variables in other assertions endorse the construction of an acyclic dependency graph for assertions. Though not shown, this graph does not contain any inconsistency. Of course, such a graph cannot prove completeness of assertions. This completeness remains a human responsibility. Nevertheless, the developed assertions can be helpful in accomplishing correct software to build a D-DBMS. Some of the required derivations cannot be specified declaratively, i.e. 'type its cardinality' (assertion 11), 'group its inherited type' (assertion 18) and 'path attribute its derived number of groups' (assertion 23). This makes it impossible to base enforcement of correctness of data distribution schemes on the interpretation of assertions only.

A practical solution might be to support a dialogue between a D-DBMS and an expert user in order to prevent incorrect specifications. In case of no-fragmentation a D-DBMS must require at least one location for the involved type. This can and should be done while the database is still empty. When instance fragmentation is appropriate, the involved instance of 'instance' can be derived by a D-DBMS from the instance of a composite type to be inserted. However, if the location of this instance is not yet specified, either this insertion must be refused or the D-DBMS must propose one or more instance locations or specify some default instance location.

In case of conditional fragmentation a D-DBMS can derive from the data model at hand which path's are legal between two types chosen by an expert user. Then, this user can choose the relevant path. According to the meta model, path attributes can be defined while the database is still empty. The inherited type of related groups can be derived from a chosen path, so it is also possible to derive which instances of instances must be accompanied by the insertion of an instance of 'instance' functioning as 'group its ref_instance'. After that registration a D-DBMS can derive which instance of 'group' must be specified. Finally, at least one related group location should be specified. At that time, the aid of an expert user can only be avoided when accepting a default group location.

According to the meta model a lot of meta data like path attributes, (referred) instances, groups and the fragment locations can be specified while the database is still empty. In that case, derivable variables like 'type its correct number of instances' should not be evaluated. Nevertheless, an early specification of meta data on fragmentation and allocation will probably cause lesser practical problems than in case of absent meta data.

- Associative data manipulation languages enable syntactically correct specification of fragments. However, a lot of these specifications are not relevant for fragmentation or difficult to check for correctness. Contrary to the Xplain language, SQL does not avoid connection traps.
- A generic approach to the enforcement of correctness of data distribution schemes has been developed. It is based on a meta model for fragmentation and allocation which avoids irrelevant fragmentation and on a set of additional restrictions.
- An unnecessary dependency between fragmentation and allocation is avoided by the new approach.

Distribution design for homogeneous distributed database systems has been discussed. Other architectures (Ram and Chastain, 1989), heterogeneous systems and the integration of existing systems (Ceri et al., 1987) have not been discussed. It is shown that fragmentation can be based on semantic links related to the structure of an organization. In this way, the design of data distribution from scratch, can be supported.

However, data distribution cannot be based automatically on semantic links to data representing organizational units because data can be related to a single organizational unit. Another problem occurs when a semantic link is completely absent. Apparently, fragmentation and allocation also require some reasonable assumptions on the intensity of data processing at different nodes. Further, the usability and the extensibility of the present approach must be further examined. Some research questions are:

- Are schemes for data distribution modifiable if modification of allocation and/or fragmentation is desired, or if a data model is modified? Is enforcement of correctness of schemes for data distribution still possible in these cases?
- Can assertions be modelled in such a way that a D-DBMS can derive when and which assertions have to be evaluated?
- Can the meta model be expanded with concepts for monitoring the use of data at different nodes, is it possible to optimize data processing?
- Can a D-DBMS decide on reallocation or replication based on this monitoring, or should this be done by an expert user?
- If additional concepts are formulated to enforce fragmentation based on composite predicates, is it then still possible to specify a sufficient set of assertions for the enforcement of correctness of data distribution schemes?

ACKNOWLEDGEMENTS

Thanks are due to the referees for their valuable comment, to my colleagues Johan ter Bekke and Dolf van der Ende for their stimulating discussions and to Koos Schonewille for drawing the figures.
REFERENCES


