Serie Research Memoranda

On Generalization in the Relational Model

M. Boogaard
R.J. Veldwijk
E.R.K. Spoor
M.V. van Dijk

Research Memorandum 1991 - 37

vrije Universiteit  amsterdam
On Generalization in the Relational Model

by

M BOOGAARD, R J VELDWIJK, E R K SPOOR and M V VAN DIJK

In the cognitive process of perceiving and structuring objects from our environment, abstraction mechanisms, i.e. classification, association, aggregation and generalization, are rudimentary. The process of structuring the object system during the development of information systems is analogous. Furthermore the structure in the relevant objects for the greater part determines the representation of data within the underlying database of an information system. The purpose of this article is to present a conceptual modelling approach for generalization structures in the relevant objects of information systems, that is based on relational principles. Although generalization is a commonly perceived construct in object systems, the relational model does not explicitly support this abstraction mechanism. The consensual approach to represent generalization structures on the basis of Codd's relational model was introduced by J.M. Smith and D.C.P. Smith in 1977. However, some coherent problems are related to their approach. These problems are mainly induced by the instability of the database structure as a result of alterations in the object system. In contrast, the proposed conceptual modelling approach consists of a highly invariant database structure, tackling most of the problems of Smith and Smith. The premise of the proposed conceptual model is formed by integration of data and metadata.

1. Introduction

Generalization is defined as an abstraction mechanism in which analogous objects are considered a generic object by deliberately ignoring individual differences between objects (see Smith and Smith 1977b). For instance, the similar objects "secretary" and "manager" can be abstracted to the generic object "employee". This abstraction mechanism is frequently used to structure objects in various environments in reality.

However, quite often the generalization structure of the objects is not so clear and recognizable as in the employee example. Moreover, clarity is influenced by the various types in which generalization structures can occur (see Brachman 1983). Consequently, the explicit diagnosis of generalization structures in object systems can easily be overlooked. Therefore information system developers need modelling approaches that explicitly support generalization.

The importance of generalization in the development of information systems has already been a subject of scientific research for a long time. In several models and approaches, for instance RM/T (see Codd 1979) and the object oriented approach (see, e.g., Meyer 1988), and in the area of artificial intelligence, generalization is considered a fundamental aspect. In current practice, many information systems are developed on the basis of the relational model (see Codd 1970) using Relational Database Management Systems (RDBMSs). However, generalization structures are not explicitly supported by the relational model and RDBMSs.

In 1977 J.M. Smith and D.C.P. Smith have introduced in two articles (1977a and 1977b) a structuring discipline which has become the consensual approach to represent generalization structures in relational databases. The purpose of their approach is to fit database design to the natural structure of the data instead of computer-related concepts. Consequently, the structure of the underlying database of an information system heavily depends on the generalization hierarchy of the objects.

However, representation of generalization structures within RDBMSs has evidently resulted in artificial data modelling for two causes. First, the modelling problems partially originate from the lack of explicitly support of generalization by both the relational model and query languages. Second, the problems are caused by the underlying modelling approach of Smith and Smith. These modelling problems are related to the necessity of a large number of application programs with the same functionality in case of many subtypes and the generally instable nature of generalization structures. In dynamic environments, such as management information systems or decision support systems, alterations in the object systems directly bring about structural adjustments of the database (see Veldwijk, Boogaard, Dijk and Spoor 1990a) because of the
one-to-one match between object and database structure. Although current RDBMSs have achieved data independence at the physical level (see Date 1990), there is no such independence on the logical level. Still, both types of data independence are among the main objectives of the relational model.

The conceptual modelling approach for generalization structures of objects introduced in this article differs from the modelling approach of Smith and Smith. This so called data dictionary model avoids one-to-one mapping between the object and database structure by integration of data and metadata. Instead of representing the generalization hierarchy in the database structure, the generalization structure is transferred to the contents of the data dictionary relations. Consequently, the resulting database structure (or data dictionary structure) is not affected by alterations in the generalization structure of the objects. Hence, the information system is logically independent as far as the generalization hierarchy of the objects is concerned.

Section 2 represents in brief the modelling approach of Smith and Smith. Furthermore, section 2 reflects the database structure resulting from their approach that applies to an example of a generalization hierarchy of objects comparable to the example used by Smith and Smith (1977b). This example illustrates and serves as a frame of reference to the subsequent sections. The problems and shortcomings originating from both the model of Smith and Smith and the relational model are elaborated in section 3. These problems can be divided into four categories, i.e. programming effort, data retrieval, integrity monitoring and system inflexibility. Section 4 introduces the proposed data dictionary model using the example of section 2 to explain the underlying concepts. The answers of the data dictionary approach to the problems described in section 3, are specified in section 5. The article concludes with some critical remarks and conclusions in section 6.

2. The Semantic Hierarchy Model

As stated in the preceding section, generalization is a common phenomenon in database design. This section briefly describes the modelling approach of J.M. Smith and D.C.P. Smith (1977a and 1977b). The model they propose, now known as the semantic hierarchy model (SHM), is based on the relational model of Codd (1970). To explain their modelling approach, consider the following example similar to the example use by Smith and Smith (1977b). The example concerns an organization with an assortment of vehicles. The assortment is extremely diversified, containing trucks, aircrafts, bicycles, boats, etc. Figure 1 shows a generalization hierarchy of the generic object "vehicle", which reflects the underlying foundation of the example.

---

![Diagram of vehicle hierarchy](image_url)

**Fig. 1. A generalization hierarchy of vehicle (see Smith and Smith 1977b)**

The generic objects, represented in Figure 1, designate classes of vehicles. A generic object is the generalization of some class of objects but it may also be the aggregation of some relationships between objects. Furthermore Figure 1 demonstrates that generic objects, for instance "rotary vehicle", "jet vehicle" and "rocket vehicle" (subtypes) can be generalized to a generic object of a higher level, "motorized vehicle" (supertype).
The vehicle hierarchy is implicitly divided into two kinds of generic objects:

1. Propulsion category.
   This kind of generic objects involves the manner in which the vehicles move forward.

2. Medium category.
   This kind of generic objects involves the main medium by which the vehicles move forward.

Therefore generic objects may have some members in common and hence do not have to be disjoint. For example, an individual airplane is motorized and flies, and is therefore an occurrence of the generic objects jet vehicle and plane. Representation of a generalization hierarchy requires that the immediately descending objects of any higher level generic object have to be ordered into groups. Each group must contain mutually exclusive classes of generic objects. A cluster (or block, see Smith and Smith 1978) is defined as a mutually exclusive group of generic objects with a common parent. A cluster must have an explicit name. The vehicle hierarchy consists of two named clusters, propulsion category (wind propelled, motorized or man powered) and medium category (land, air or water), belonging to the parent "vehicle".

Consider the following restricted assortment of vehicles of the organization at issue (see Figure 2).

<table>
<thead>
<tr>
<th>Properties \ Vehicles</th>
<th>Car</th>
<th>Airplane</th>
<th>Boat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification number</td>
<td>V1</td>
<td>V2</td>
<td>V3</td>
</tr>
<tr>
<td>Manufacturer</td>
<td>Mazda</td>
<td>Boeing</td>
<td>Aqua Co</td>
</tr>
<tr>
<td>Price</td>
<td>65.4</td>
<td>7900</td>
<td>12.2</td>
</tr>
<tr>
<td>Weight</td>
<td>10.5</td>
<td>840</td>
<td>1.9</td>
</tr>
<tr>
<td>Horse-power</td>
<td>150</td>
<td>9600</td>
<td>-</td>
</tr>
<tr>
<td>Fuel capacity</td>
<td>300</td>
<td>2600</td>
<td>-</td>
</tr>
<tr>
<td>Maximum altitude</td>
<td>-</td>
<td>30</td>
<td>-</td>
</tr>
<tr>
<td>Takeoff distance</td>
<td>-</td>
<td>1000</td>
<td>-</td>
</tr>
<tr>
<td>Number of sails</td>
<td>-</td>
<td>-</td>
<td>2</td>
</tr>
</tbody>
</table>

Fig. 2. Assortment of vehicles

Identification number, manufacturer, price and weight are properties relevant to all vehicles. However, the remaining properties horse-power, fuel capacity, maximum altitude, takeoff distance and number of sails are properties relevant to several but not all vehicles. The relevant properties of an individual vehicle will differ from class to class. Another feature of a generalization hierarchy is that the lower the generic level of the class in which the object is considered, the more relevant properties an individual object will have. The reason is that an object inherits all properties of the higher level classes of objects to which the object belongs.

Smith and Smith create a database relation for each generic object in the generalization hierarchy. Each relation has the following structure (see Figure 3), where $A_1, \ldots, A_n$ are the attributes common to all the descending objects of the generic object (i.e. component attributes, see Smith and Smith 1978) and $C_1, \ldots, C_m$ are the names of the clusters belonging to the generic object (i.e. category attributes, see Smith and Smith 1978).

Fig. 3. The generic structure of a relation
This above approach results in the following four database relations, reflecting a part of the complete database structure (see Figure 4):

<table>
<thead>
<tr>
<th>VEHICLE</th>
<th>ID#</th>
<th>MANUFACT</th>
<th>PRICE</th>
<th>WEIGHT</th>
<th>PROP_CAT</th>
<th>MED_CAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>Mazda</td>
<td>65.4</td>
<td>10.5</td>
<td>MOT_VEHICLE</td>
<td>LAND_VEHICLE</td>
<td></td>
</tr>
<tr>
<td>V2</td>
<td>Boeing</td>
<td>7900</td>
<td>840</td>
<td>MOT_VEHICLE</td>
<td>AIR_VEHICLE</td>
<td></td>
</tr>
<tr>
<td>V3</td>
<td>Aqua Co</td>
<td>12.2</td>
<td>1.9</td>
<td>WING_VEHICLE</td>
<td>WATER_VEHICLE</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MOT_VEHICLE</th>
<th>ID#</th>
<th>MANUFACT</th>
<th>PRICE</th>
<th>WEIGHT</th>
<th>H_POWER</th>
<th>FUEL_CAP</th>
<th>MOT_CAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>Mazda</td>
<td>65.4</td>
<td>10.5</td>
<td>50</td>
<td>200</td>
<td>ROTARY_VEHICLE</td>
<td></td>
</tr>
<tr>
<td>V2</td>
<td>Boeing</td>
<td>7900</td>
<td>840</td>
<td>9600</td>
<td>2600</td>
<td>JET_VEHICLE</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AIR_VEHICLE</th>
<th>ID#</th>
<th>MANUFACT</th>
<th>PRICE</th>
<th>WEIGHT</th>
<th>MAX_ALT</th>
<th>TAKE_DIST</th>
<th>LIFT_CAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>V2</td>
<td>Boeing</td>
<td>7900</td>
<td>640</td>
<td>30</td>
<td>1000</td>
<td>PLANE</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WIND_VEHICLE</th>
<th>ID#</th>
<th>MANUFACT</th>
<th>PRICE</th>
<th>WEIGHT</th>
<th>NUMB_SAILS</th>
</tr>
</thead>
<tbody>
<tr>
<td>V3</td>
<td>Aqua Co</td>
<td>12.2</td>
<td>1.9</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 4. Part of the database

One of the consequences of the SHM is that relation names appear as domain values of the category attributes. For instance, the attributes PROP_CAT and MED_CAT in the relation VEHICLE, defined on the domains "propulsion category" and "medium category" respectively, contain among others the relation names MOT_VEHICLE, AIR_VEHICLE, and WIND_VEHICLE as values (the other names are not mentioned as a relation in Figure 4).

The domain in a relation which contains names of descending relations is called an image domain for the descendants. For example, the domain "propulsion category" of attribute PROP_CAT is the image domain for the descendant relations WIND_VEHICLE, MOT_VEHICLE, and MAN_VEHICLE. Notice that there is a one-to-one mapping between clusters in a generalization hierarchy and image domains in its relational reproduction. In conclusion, category attributes are always defined on domains which are image domains for the descendants.

As stated before, the component attributes of a generic object are inherited by its descendants. Obviously, this results in redundant information in the database relations. Smith and Smith allow redundancy in a relational hierarchy implementation when the consistency of the redundant information can be guaranteed. Therefore they extend the two integrity rules of the relational model (i.e. entity and referential integrity) with three additional relational invariants, namely:
1. Each tuple in an immediate descending relation has one parent tuple in the generic relation.
2. Each tuple in a generic relation has one child tuple in an immediate descending relation which is identified by the value of the category attribute (in case of a not null value situation).
3. No tuple in a generic relation has more than one child tuple in immediate descending relations in the same cluster (mutually exclusive).

The basic two relational integrity rules are necessary to represent a set of well-defined relations (as a collection of aggregate objects). The additional three are necessary to represent the generalization structure in an appropriate manner. The result of the three additional rules is a one-to-one relationship between a supertype relation and a subtype relation. Every relation in the database must comply with these five integrity rules before and after database insert, delete and update operations.
3. Coherent Problems

This section describes the modelling problems that originate from both the SHM and the relational model. Despite of the benefits of the SHM this model brings about several problems. These problems are caused by the fact that the relational model and first order query languages do not explicitly support SHM\(^1\) and furthermore by the shortcomings of SHM itself. The purpose of this section is to address the problems without explicit assignment to one of the described origins. The problems are subdivided into four categories:

1. Programming effort.
2. Data retrieval.
3. Integrity monitoring.
4. System inflexibility.

There is a certain degree of interdependence between these categories. Nevertheless the subdivision is made to emphasize the different features of the categories.

3.1. Programming effort

In the SHM each generic object is represented by a relation. Almost every relation requires specific application programs for several analogous functions. For instance, each relation demands a suitable input application program in order to obtain information for all relevant attributes. Thus, an input application program for the relation MOT VEHICLE must include the attributes ID#, MANUFACT, PRICE, WEIGHT, HPower, FUEL CAP, and MOT CAT, while another input application program is required for the relation AIR VEHICLE including the attributes ID#, MANUFACT, PRICE, WEIGHT, MAX ALT, TAKE DIST, and LIFT CAT. Each application program must be coded separately for the specific relation thus resulting in significant programming effort. Consequently, the information system contains a large number of application programs with comparable functionality whenever the underlying generalization hierarchy consists of many subtypes.

Obviously, the large amount of application programs increases the complexity and reduces the maintainability of the information system.

3.2. Data retrieval

Generally, retrieval of data is difficult without complete knowledge of the database structure. In the case of generalization, this problem is even bigger because knowledge of the generalization structure within the database is required too. When there is no comprehensive overview of the database structure and its interweaved structures a query over the database must be done in multiple steps. Consequently, questions result in multiple queries while analogous questions in a situation without generalization require only one query. For instance, the question "Retrieve all available information of the individual vehicle 'V2'" includes several queries. First, a query on VEHICLE (the root relation) is required to identify the subtypes to which 'V2' belongs. The result are the names of the relations that represent the vehicle subtypes containing 'V2' as an individual object. Second, several queries, containing the identified relation name(s), are required to search through the hierarchy and finally retrieve the information at the lowest level (the leaf relations) for each cluster. Consequently, the composer of the queries must be precisely aware of which attributes of a relation are component attributes and which are category attributes (image domains).

The queries necessary for the described question, illustrated with SQL-statements, are depicted in Figure 5.

---

\(^1\) A precise description of the influence of this deficiency on the problems is beyond the scope of this article.
According to Smith and Smith (1977b) "a first order language appears to be adequate for all relational models no matter how many levels of generalization they contain" because of the uniform handling of all objects. Nevertheless, they primarily propose a higher order language to facilitate stability of application programs (see "System inflexibility" below), but this language can also be used to compose single queries. In this higher order language all clauses of a query can consist of variables when relation names and attribute names are required. The current SQL standard, usually present in contemporary RDBMSs, does not support this facility. The same result can be realized using a dynamic query language. However, this is a surrogate higher order language. Thus, relatively simple information needs result in rather complicated queries.

A totally different problem is a query involving attributes relevant to several, but not all, classes of vehicles. For instance, assume that the generic object AIR_VEHICLE has no the property price. Consequently, PRICE is no longer a component attribute and thus no longer shown in the root relation VEHICLE. A question such as "Retrieve the identification numbers and manufacturers of individual vehicles with price greater than 100" results in the following query including every subtype relation of the generalization hierarchy except AIR_VEHICLE (see Figure 6).
select ID#, MANUFACT
from MOT_VEHICLE
where PRICE > 100
union
select ID#, MANUFACT
from WIND_VEHICLE
where PRICE > 100
union
... etc ...
Results in: ID# MANUFACT
V2 Boeing
... ********

Fig. 6. Query involving attributes of different object types

The query of Figure 6 looks quite simple but there are three aspects to take into consideration. First, the query has a structural component that must be previously known by the composer (i.e. the FROM-clauses). The composer must precisely be aware of the current structure of the underlying data model in order to retrieve the desired information. In this case, the composer has to know the database relations that contain PRICE as an attribute.

The second aspect is the growing complexity. The more database relations are joined in a query, the more complex queries become. The complexity of a query depends on the number of database relations involved and the specific query conditions stipulated. In the above query these two aspects are limited to only a few relations and one simple condition which results in a rather simple query. However, in a generalization structure with a large number of subtype relations complex queries are inevitable.

The last aspect involves the inevitable hard-coding of the query in application programs resulting in application programs that are time-dependent (see "System inflexibility" below).

3.3. Integrity monitoring
To ensure the integrity of the information system Smith and Smith propose five integrity rules. The three additional integrity constraints form no part of the relational model and are therefore not supported by any contemporary RDBMS. Ideally, integrity rules must be automatically assured by the RDBMS itself and not by the application programs (see Date 1985). The current available RDBMSs support at most the two relational integrity rules, i.e. entity and referential integrity and hence user-defined or additional constraints must be designated separately. Consequently, the three remaining integrity rules must be specified for each relation in the constraint support facility of the RDBMS (if available), in the application programs or in triggered procedures. In order to ensure the integrity of the database, Smith and Smith (1977b) described a method for correcting actions for each integrity violation per database operation, i.e. insert, delete or update.

3.4. System inflexibility
The SHM explicitly represents the underlying generalization structure of the information system. According to Smith and Smith a relational implementation of the generalization structure requires mapping of the specific generalization hierarchy to the database structure. The consequence of this structural mapping is instability of the database in case of alterations of the generalization structure\(^2\). For instance, the database structure is extended with one relation in addition to the extension of the generalization hierarchy with the new generic object "amphibious vehicle" within the medium category, including the four common attributes and one specific

\(^2\) Normally, instability of data models is not considered a problem. The fiction exists that a data oriented modelling approach results in a stationary data model.
attribute, DRAUGHT\(^3\).

However, instability of database structures affects existing application programs which must be adjusted to every alteration of the database structure (see Veldwijk, Boogaard, Dijk and Spoor 1990a). Although a first order query language is adequate to retrieve and manipulate data it results in hard-coding of the queries with comparable functionality for different relations in application programs. Depending on the value of a variable after a query, other queries will be executed by an if-statement or case-statement (see Figure 5 and 6). Because application programs depend on the generalization structure on one specific moment (in other words, the application programs are time-dependent) alterations of the generalization structure not only affect the database structure but these application programs and certain constraints too. To introduce stability of application programs Smith and Smith (1977b) propose a higher order language. However, current query languages do not support higher order facilities.

Furthermore, several application programs and constraints must be added to the system. In general, the structure of the database does not remain the same after an alteration of the underlying generalization structure. Moreover, a structural alteration of the database also affects application programs and constraints.

The generalization structure in the "vehicle" environment is an example of a generally stable environment. This might be an argument to implement the generalization structure in the way Smith and Smith propose (the problems described above still apply). However, generalization is a common structure in dynamic environments like management information systems and decision support systems. These environments are characterized by frequently changing information needs. Most evolving information needs require database structure alterations and, consequently, maintenance of application programs.

In summary, the SHM is not completely satisfying whenever the number of subtypes is substantial, the generalization hierarchy is dynamic and specializing attributes apply to multiple subtypes without major extensions of both the relational model and query languages.

4. The Data Dictionary Model

The problems described in the previous section result from the lack of self-knowledge of the system. Because the database system is not "aware" of its own structure, it is not capable to adjust to evolving information needs of the environment. Furthermore, the lack of self-knowledge leads to an increased complexity of the system in case of data retrieval and integrity monitoring. The logical conclusion is that a system must contain data about the data it keeps, i.e. metadata. A system containing this kind of structural information is a data dictionary.

Ross (1981) stated that a data dictionary contains information about the structure and the contents of the database, and that a data dictionary itself is a database, too. A further deduction is that the data dictionary thus can contain its own structural definition (and becomes self-referential, see Ross 1981). With the degree of self-knowledge thus obtained all kinds of structural alterations of the database are possible with a minimum of disruption to the functions of the system (see Veldwijk, Boogaard, Dijk and Spoor 1990a).

The principle of the so called data dictionary approach is integration of data and metadata. The result is that nothing, or almost nothing\(^4\), needs to be hard-coded into the system, that is, hard-coding can be avoided in the application programs as well as in the structure of the database. The absence of hard-coding in both the application programs and the database structure results in inherently flexible information systems. Information systems based on a stable

\(^3\) In our example "amphibious vehicle" is an immediate descendant of "vehicle" because of the constraint that a group consists of mutually exclusive objects. So it is not an example of bad modelling (see Smith and Smith, 1978).

\(^4\) There will always remain a hard-coded entry-point in an information system because first of all it must execute a determined startup procedure. Consequently, the kernel of the data dictionary must be invariant (see Ross 1981, but also Hofstadter 1979 for a more general point of view).
database structure do have the built-in ability to adjust themselves to a change of structure in their object systems without affecting the application programs. The application programs are generalized and immune to many changes in the database structure. The explanation for this immunity is that the application programs run against the data dictionary which contains the information about the current structure of the database. Furthermore, in this situation it is not necessary to develop specific application programs with comparable functionality. Instead, generalized application programs can be used.

The above description may seem a bit abstract but in principle the data dictionary strategy proposes application of an approved method on a higher level of abstraction. It is common practice to increase the flexibility of a system by explicitly storing data in a database instead of using traditional data files which have hard-coded structures in the application programs. For instance, it is unusual to hard-code the VAT-percentage in the application programs. Nowadays, the VAT-percentage is stored in a database and is read from the database by the application programs. Thus a modification of the VAT-percentage from 4 to 3.5 or modifications of the physical structure of the stored data do not require recoding of the application programs (i.e. physical data independence, see Date 1990). Comparably, it is possible to gather information about the structure of the database itself from the data dictionary.

However, the important conclusion that the structural definition of the database is equivalent to the contents of the data dictionary is usually overlooked. This conclusion implies that alterations of the database structure realized by DDL-statements can also be accomplished by DML-statements on the contents of the data dictionary (see Lek and Buitendijk 1991). Application programs which run against the data dictionary are immune to mutations of the contents of the data dictionary (i.e. logical data independence, see Date 1990), as the above application programs are immune to changes in VAT-percentages. The data dictionary model integrates the data dictionary into the database and does not consider the data dictionary to be a passively storage device of the metadata.

Smith and Smith (1978) already conclude that the structural definition of the database (and thus the generalization structure) can be stored in a data dictionary. Furthermore, they state that it is possible to integrate the data dictionary into the conceptual model. This allows queries on and manipulation of the data dictionary. However, they did not elaborate the application of the data dictionary to generalization structures. Note that in the SHM the concept of image domains is a provisional reflection of the structural database definition. Image domains contain relation names that represent the generalization structure.

Implementing the data dictionary approach for the generalization structure described in section 2, results in the following data model.

![Diagram](image)

**Fig. 7. The data dictionary model**

The meta-entity-types shown in Figure 7 represent the following data dictionary relations:
1. **CLUSTER.**
   The meta-entity-type CLUSTER represents the clusters which occur in the generalization structure. The description of CLUSTER is:

   **CLUSTER (CL_NAME, OBT#)**

   Where: CL_NAME = The name of a cluster.
   OBT# = The code of the object type.

2. **OBJECT_TYPE.**
   The meta-entity-type OBJECT_TYPE represents the types or classes of objects which occur in the generalization structure. The description of OBJECT_TYPE is:

   **OBJECT_TYPE (OBT#, CL_NAME, OBT_NAME)**

   Where: OBT# = The code of an object type.
   CL_NAME = The name of the cluster to which the object type belongs.
   OBT_NAME = The name of the object type.

3. **MEMBERSHIP.**
   The meta-entity-type MEMBERSHIP represents the many-to-many relationship between individual objects and object types. Consequently, MEMBERSHIP represents the object type(s) to which an individual object belongs. The description of MEMBERSHIP is:

   **MEMBERSHIP (OBt, OBT#)**

   Where: OBt = The surrogate object identifier of an individual object. The purpose of OBt is to identify the existence of an individual object inside the data dictionary model. Further, OBt is a system generated object identifier and does not have an intrinsic meaning (see Codd 1979). OBt is necessary because the essence of the objects, i.e. its properties, is described in the data dictionary relations ATTRIBUTE and VALUE. However, the user still identifies an individual object by a user key like identification number and will never be confronted with surrogate values.
   OBT# = The code of the object type to which the individual object belongs.

4. **OBJECT.**
   The meta-entity-type OBJECT represents the set of surrogate object identifiers of the individual objects. Hence, OBJECT is fully comparable with an E-relation in RM/T (see Codd 1979). The description of OBJECT is:

   **OBJECT (OBt)**

   Where: OBt = The surrogate object identifier of an individual object.

5. **ATTRIBUTE.**
   The meta-entity-type ATTRIBUTE represents the domains on which the attributes of the object types are defined. The description of ATTRIBUTE is:

   **ATTRIBUTE (OBT#, DOMAIN#)**

   Where: OBT# = The code of the object type to which the attribute belongs.
   DOMAIN# = The code of the domain to which the attribute belongs.

   Whenever more attributes within one object type are defined on the same domain, a role identifier must be included in the primary key of ATTRIBUTE instead of DOMAIN#. Consequently, the role identifier becomes also a part of the foreign key in VALUE instead of DOMAIN# (see "7. VALUE" below).
6. **DOMAIN.**

The meta-entity-type **DOMAIN** represents attributes that can be joined meaningfully (and thus attributes having the same pool of values). The description of **DOMAIN** is:

**DOMAIN** (DOMAIN#, DOMAIN_NAME, DATATYPE, WIDTH, SCALE)

Where:
- **DOMAIN#** = The code of a domain.
- **DOMAIN NAME** = The name of the domain.
- **DATATYPE** = The data type of the values in the domain (i.e. C, N or D).
- **WIDTH** = The width of the values in the domain.
- **SCALE** = The scale of the values in the domain.

7. **VALUE.**

The meta-entity-type **VALUE** represents the value of an attribute of an individual object. The description of **VALUE** is:

**VALUE** (OBJ#, OBT#, DOMAIN#, ATT_VALUE)

Where:
- **OBJ#** = The surrogate object identifier of an individual object.
- **OBT#** = The code of the object type to which the individual object belongs.
- **DOMAIN#** = The code of the domain to which the attribute value belongs.
- **ATT_VALUE** = The value of the attribute.

The above described relations of the data dictionary model contain the following data for the "vehicle" example (see Figure 8).

---

5 As far as implementation is concerned the values of the column ATT_VALUE are variable length, alphanumeric fields. The format descriptions of the values, like for instance data type, are stored in the data dictionary relation **DOMAIN**.
Like in the SHM, a lot of the information is redundant, predominantly in the VALUE relation. As stated before, the redundancy is a result of the inheritance of properties. There is no way to eliminate this logical redundancy without extending the first order query languages or introducing new first order facilities. A first order query can not include more than two levels of the generalization hierarchy without repeating common attributes and their values for every individual object. The reason for this restriction is that the generalization hierarchy is structured
by a recursive relationship and first order query languages do not include a recursive join facility (see, e.g., Date 1990 and Codd 1990). A regular equi-join between a supertype and a subtype covers just two levels of the hierarchy\(^6\). However, this redundancy is acceptable because the consistency of the redundant information is implicitly guaranteed. The consistency of values of common attributes is assured because manipulation of the value requires just one action on one relation, namely VALUE. For instance, an update of the weight of the individual object 'VI' from '10.5' to '11' requires only one update statement without any influence on the consistency (see Figure 9). Note that an analogous update on a relational implementation of the SHM requires multiple actions.

\[
\text{update VALUE}
\text{set ATT VALUE = '11'}
\text{where DOMAIN# = 'WEIGHT'}
\text{and OBC =}
\text{(select distinct OBC from VALUE}
\text{where DOMAIN# = 'ID#'
\text{and ATT VALUE = 'VI'})}
\]

Fig. 9. An update query on a redundant attribute

As depicted in Figure 7, there are two relationships between the data dictionary relations OBJECT_TYPE and CLUSTER. First, OBJECT_TYPE and CLUSTER are related by the foreign key OBT# of CLUSTER. Second, OBJECT_TYPE and CLUSTER are related by the foreign key CL_NAME of OBJECT_TYPE. These two relationships form the described generalization structure. Note that the generalization structure is no longer explicitly represented but instead transferred to the contents of the data dictionary relations OBJECT_TYPE and CLUSTER. However, it can be made explicit by means of an outer-join\(^7\) between OBJECT_TYPE and CLUSTER over CL_NAME (see Figure 10). Note that a relational implementation of the SHM does not provide the required information to represent the generalization structure in this way.

\[
\text{select OBJECT TYPE.OBT#, CLUSTER.OBT#}
\text{from OBJECT TYPE, CLUSTER}
\text{where OBJECT_TYPE.CL_NAME = CLUSTER.CL_NAME (+)}
\]

Results in:

\[
\begin{array}{ll}
\text{OBJECT TYPE.OBT#} & \text{CLUSTER.OBT#} \\
\hline
\text{VEHICLE} & \text{VEHICLE} \\
\text{WIND VEHICLE} & \text{VEHICLE} \\
\text{NOV VEHICLE} & \text{VEHICLE} \\
\text{MAR VEHICLE} & \text{VEHICLE} \\
\text{LAND VEHICLE} & \text{VEHICLE} \\
\text{AIR VEHICLE} & \text{VEHICLE} \\
\text{WATER VEHICLE} & \text{VEHICLE} \\
\text{ROTARY VEHICLE} & \text{VEHICLE} \\
\text{JET VEHICLE} & \text{VEHICLE} \\
\text{ROCKET VEHICLE} & \text{VEHICLE} \\
\text{FLANE} & \text{VEHICLE} \\
\text{HELICOPTER} & \text{VEHICLE} \\
\end{array}
\]

Fig. 10. The generalization structure represented by means of an outer-join

The interpretation of the result of the outer-join is that the object represented by the value of

---

\(^6\) The acyclic graph structure of hierarchies requires a recursive join. Codd announced the publication of a complete solution of this recursive join problem while introducing RM/V3 (see Codd 1990).

\(^7\) The outer join is realized by adding "(+)". This symbol represents the specific outer-join for the database management system ORACLE in which the example has been implemented.
OBT# of OBJECT_TYPE is the immediate descendant of the object represented by the value of OBT# of CLUSTER. The root of the generalization hierarchy, 'VEHICLE', can be recognized by the absence of a value for OBT# of CLUSTER resulting from the outer-join. Furthermore, the two relationships between OBJECT_TYPE and CLUSTER indicate to which object types the clusters belong and represent the permitted clusters in the generalization structure.

Although it is obvious that the data dictionary approach described in this section has repercussions on the performance of the information system, it should be noted that the aim is to present a conceptual framework and thus performance is not a fundamental aspect. In section 6, however, we will present an implementation alternative of the proposed data dictionary approach together with the consequences for performance.

5. Problem Solutions

This section discusses the solutions of the data dictionary model approach to the four types of problems of the SHM as described in section 3. As stated before these problems are not only caused by the SHM but are also influenced by the restrictions of the relational model and first order query languages.

5.1. Programming effort
The data dictionary approach permits the development of extremely generalized application programs for several analogous functions. Instead of hard-coding every relation and its attributes in individual application programs, the required information can be read from the data dictionary relations. This implies that it is no longer necessary to develop application programs for every similar function for each relation in the database separately. Referring to the two input application programs mentioned in section 3, one generic input application program can be developed using the data dictionary relation ATTRIBUTE to obtain the relevant attributes of the objects MOT_VEHICLE and AIR_VEHICLE.

Furthermore, this gain in programming effort is not limited to one information system. With some minor adjustments, the application programs can also be used in other information systems. Finally, the smaller amount of application programs in an information system reduces the complexity and increases the maintainability of the information system in general.

5.2. Data retrieval
The main advantages of the data dictionary model for data retrieval, are that the composer of the queries no longer needs to have complete knowledge of the generalization structure within the database, and that questions can be represented by single queries. These two advantages can be illustrated by repeating the queries of section 3 (see Figure 5 and 6) adjusted to the data dictionary model (see Figure 11).

---

8 In order to represent the relevant attributes in a desired layout, the data dictionary relation ATTRIBUTE must be extended with information about the place of an attribute on the screen. Consequently, alterations of the layout of a screen require no modification of the application programs. The result is a rise of flexibility of the information system.
1. select distinct DOMAIN#, ATT_VALUE
   from VALUE
   where OBC in
     (select OBC
      from VALUE
      where DOMAIN# = 'ID#'
      and ATT_VALUE = 'V2')

Results in: DOMIAN# ATT_VALUE
--------------
ID#       V2
MANUFACT Boeing
PRICE     7,900
WEIGHT    840
N_POWER   9600
FUEL_CAP  2600
MAX_ALT   38
TAKDIST   1000
........   .......... (and other DOMAIN# ATT_VALUE combinations, see Figure 5)

2. select distinct DOMAIN#, ATT_VALUE
   from VALUE
   where (DOMAIN# = 'ID#'
   or DOMAIN# = 'MANUFACT')
   and OBC in
     (select OBC
      from VALUE
      where DOMAIN# = 'PRICE'
      and to_number(ATT_VALUE) > 100)
   order by OBC:

Results in: DOMIAN# ATT_VALUE
--------------
ID#       V2
MANUFACT Boeing
........   .......... (etc., see Figure 6)

Fig. 11. Queries on the data dictionary model

Although the queries look complex, the specific information need does not require multiple
queries. The queries no longer contain the names of the relations that represent the
generalization structure. Hence, composition of the queries does not require complete knowledge
of the generalization structure in the database. Note that the first query does not result into
repetition of attributes and their values (compare Figure 5).

Figure 11 also illustrates that the representation of the retrieved data leaves much to be
desired. However, the representation can easily be improved by a generally applicable procedure
representing the data in a normal outline.

5.3 Integrity monitoring

As stated in section 2, the integrity monitoring of an information system structured by the SHM
requires five relational invariants. Besides the two general relational integrity rules, i.e.
referential and entity integrity, Smith and Smith include three additional relational invariants.
The data dictionary approach implicitly ensures two of the additional relational invariants of the
SHM by means of the two general relational integrity rules. The first additional integrity rule of
Smith and Smith, "each tuple in an immediate descending relation has one parent tuple in the
generic relation", is superfluous because generic and subtype relations no longer occur. In the
data dictionary model, the generalization hierarchy is no longer represented by supertype and
subtype relations. Instead it is represented by the two foreign key to primary key relationships
between OBJECT TYPE and CLUSTER. The integrity of these two relations is simply
 guaranteed by the referential integrity rule.

A similar argumentation applies, "each tuple in a generic relation has one child tuple in an
immediate descending relation which is identified by the value of the category attribute (in case of
not null situation)", to the second additional integrity rule.

In contrast, the third additional integrity rule, "no tuple in a generic relation has more than one
child tuple in immediate descending relations in the same cluster (mutually exclusive)", is not
implicitly ensured and therefore has to remain an explicitly defined constraint.
However, the data dictionary model needs an additional constraint. The value of ATT_VALUE of VALUE must be consistent with the format description in DOMAIN of the domain on which the attribute is defined. Consequently, integrity monitoring of the data dictionary model is comparable with integrity monitoring of the SHM.

5.4. System inflexibility
The data dictionary approach results in an information system that has a certain degree of inherent flexibility. In the data dictionary model the generalization structure does not determine the underlying database structure. Instead the generalization hierarchy is reflected in the contents of the data dictionary relations. The transfer from database structure determinant to data dictionary contents determinant result in an underlying database structure that remains stable during alterations in the generalization hierarchy. For example, the alteration of the "vehicle" generalization structure with the addition of the generic object "amphibious vehicle" within the medium category (including the four common attributes and one specific attribute, DRAUGHT) no longer leads to an extension of the database structure with one relation (see section 3). In the data dictionary model such an alteration results in an insert into OBJECTTYPE, ATTRIBUTE and DOMAIN. Alteration of the generalization hierarchy results in a change of the contents of the data dictionary relations instead of a change in the database structure.

Because the application programs run against the data dictionary relations, alterations of the generalization structure directly influence the application programs. The absence of hard-coding results in queries and application programs that are time-independent as far as modifications of the generalization structure are concerned. In both queries on the data dictionary model the hard-coding elements are eliminated (see figure 11).

In conclusion, the data dictionary model forms the basis of the development of inherently flexible information systems. In other words, with the data dictionary model a higher level of logical data independence can be achieved as far as this kind of generalization is concerned.

6. Conclusions
The two modelling approaches considered, the semantic hierarchy model (SHM) and the data dictionary model, result in different views on the problems concerning generalization because the principles of both conceptual models differ. The purpose of the SHM is to present a structuring discipline that corresponds with the natural structure of data. The outcome of this approach is the explicit representation of the generalization hierarchy by the database structure. But the described problems of the approach of Smith and Smith mainly depend on the hard-coding both in the application programs and in the database structure, that is directly caused by this one-to-one mapping between the generalization structure and the database structure.

The proposed modelling approach, on the other hand, is originally aimed at the development of inherently flexible information systems. As stated before the independence on the logical level as far as the generalization structure is concerned, is obtained by avoiding hard-coding in both application programs and database structure. This results in a large degree of flexibility. In this section we discuss several critical aspects of the proposed data dictionary model. Both conceptual models are influenced by the restrictions of the relational model and first order query languages. Hence, several described problems are partially caused by these restrictions. Furthermore, we will indicate some relationships with current topics in information system development and point out some directions for further research based on the integration of data and metadata.

In section 4 we have mentioned performance. Although it is possible to implement the data dictionary model straightforward (using a dynamic query language to deal with the described

\[ ^9 \] Still, the data dictionary approach opens some further perspectives towards constraints support. In section 6 we will briefly consider the subject of constraints support.
problems), in most cases the consequences for performance will be unacceptable. The alternative is to increase performance by introducing controlled redundancy by a combination of the data dictionary and the SHM. The mixture of the two approaches results in an integration of data dictionary relations and database relations. Instead of using the VALUE-relation, each object type is represented by a "normal" database relation containing the attribute values of the specific object type. Because the data dictionary relations hold the structural definitions of these database relations, too, this information is redundant. This redundancy, however, is controlled because a required alteration of the database structure due to a modification of the data dictionary relations, can be executed automatically. Consequently, the advantage of the data dictionary approach concerning the programming effort is lost, because the combination of approaches requires supplementary application programs. These application programs can be subdivided in two categories, namely:

1. An application program necessary to adjust the database relations according to an alteration of the data dictionary relations.
2. An application program to translate queries based on the data dictionary approach into queries relevant for the SHM and vice versa (see Veldwijk, Boogaard, Dijk and Spoor 1990a).

In conclusion, the gain of performance is at the expense of the programming effort but the advantage of inherent flexibility remains the same. Besides that, the additional application programs are reusable because they, too, operate on the data dictionary.

Another critical aspect of the data dictionary approach is the situation of multiple generalization structures in the relevant objects of an information system. In that case, each generalization structure requires a distinct data dictionary structure because the described data dictionary can not represent interrelated generalization structures or deal with multiple inheritance.

Nevertheless, we believe that it is possible to develop a more general data dictionary approach in which multiple generalization structures of an information system can be stored. The description of this general method, however, is beyond the scope of this article and requires further research.

As stated before, the data dictionary approach is a conceptual solution to generalization problems, based on relational principles. This does not imply that further research on generalization in other fields of automation, for instance in the object oriented or artificial intelligence area, becomes irrelevant. Instead, the joint fields of interest indicate to make a combination of the various research areas and the role the data dictionary approach might fulfill in the different areas. Moreover, some characterizing elements of object orientation, like inheritance and reusability, are also aspects of the data dictionary approach. Although artificial intelligence is primarily concerned with generalization, the other abstraction mechanisms, i.e. classification, association and aggregation, are also important. We are convinced that the data dictionary model, with some extensions, can contribute to many of these aspects of different areas, including the other abstraction mechanisms.

In conclusion, generalization must be considered a fundamental aspect in system development, but it is not supported by the relational model. Accordingly, extensions of the relational model and first order query languages are required to support various kinds of generalization structures in an appropriate manner using one of the conceptual models. In our opinion, the data dictionary model is preferable because it requires less extensions than the SHM.

In this article we have only described the possibilities to solve problems caused by generalization hierarchies. But the approach offers more perspectives. Without listing a complete set, we illustrate these further prospects with two brief characterizations which are particularly relevant to the subject of this article. First, an extended data dictionary model contains sufficient information to automate a part of the maintenance. It is possible to implement alterations in the database structure and adapt the affected queries in application programs automatically (see Veldwijk, Boogaard, Dijk and Spoor 1990a). A second challenge is to include a constraints

---

10 A description of an implementation, based on the data dictionary model, in a financial environment can be found in Veldwijk, Dijk, Boogaard and Spoor (1990b).
support facility. This facility enables the one-off definition of constraints applicable for every relation in the database, i.e., metaconstraints. The metaconstraints also involve constraints other than the referential and entity integrity rule. Furthermore, the constraint support facility introduces a flexible approach to the definition of other constraints. The constraints are no longer hard-coded in the application programs or in triggered procedures. Instead, the data dictionary contains and controls the constraints and guarantees the integrity (see Date 1985).
References


About the authors