Product modelling using multiple levels of abstraction: instances as types

ELSEVIER

Computers in Industry 24 (1994) 17–28

COMPUTERS IN INDUSTRY

Product modelling using multiple levels of abstraction Instances as types

Frederik Erens [a,b,*], Alison McKay [c], Susan Bloor [c]

[a] *Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands*
[b] *KPMG Lighthouse, P.O. Box 6427, 5600 HK Eindhoven, The Netherlands*
[c] *Department of Mechanical Engineering, University of Leeds, Leeds LS2 9JT, UK*

(Received August 10, 1993; accepted November 8, 1993)

Abstract

Most databases make use of three levels of abstraction, namely: the data dictionary, the database schema and the database contents. The data dictionary describes the structure of the database schema whilst the database schema describes the structure of the database contents. This approach fits perfectly in situations with large quantities of "simple" data and relatively small and stable structures. In this paper, we will focus on "product models" which cannot be modelled easily with these levels of abstraction. We will illustrate the modelling problem with an example and present a solution using the Leeds Product Data Editor.

1. Problem statement

The use of databases is possibly as old as writing. Both merchants and armies created lists of formatted data, more than 2000 years ago. Once the list (database) structure was defined, data could be added to the list by filling in the necessary attributes. This century the use of formatted data has increased dramatically, especially in governmental organizations and large companies. Information about citizens, for example, is stored in tens of databases with hundreds of attributes describing specific characteristics of these citizens. Company-owned databases for customers, suppliers and employees add even more attributes to this list. All these databases have a relatively stable database schema in that the structure of the data that is stored will not change considerably over time. This stability is accompanied by large quantities of repetitive data, often thousands or millions of occurrences of a certain structure. Computer-based information systems that are built on top of these databases are especially good in processing these large quantities of data.

The last decade, however, has shown a considerable growth of nontraditional applications, like personal information management systems, computer-aided co-operative work systems and computer-aided engineering systems [1]. Features that

* Corresponding author.

0166-3615/94/$07.00
SSDI 0166-3615(93)E0038-D

best characterize this class of specialized applications are:

- Their structural data (data description, data interrelation and data classification) is typically more dynamic than conventional data-intensive applications and is intertwined with, and often a part of, the repetitive data itself.
- The amount of structural information is large compared to the simple information content, for example the geometry description of a truck is highly structured and includes a relatively small amount of repetitive data.
- End-users have different views on the same data. Some are working on product family structures, others on detailed component descriptions, all using their own terminology and applications. What they have in common is that the users are strongly connected with the data. Browsing the database seeing the structure as well as the repetitive data is an important aspect of their work. Traditional business applications have rigid user interfaces to shelter the user from the structure of the database itself.

Applications where structural data is intertwined with repetitive data often encounter a modelling problem when traditional database technology is applied. Design choices must be made about which part of the data is seen as structural and modelled in the database schema, and which part of the data is seen as repetitive and modelled in the database as occurrences of the database schema. However, a closer look at these applications reveals that there is a sliding scale between structural data and repetitive data. It is not always possible to split the data into two distinctive groups. This statement is elucidated in the next section with an example of developing product families.

The remainder of this paper is organized as follows. In Section 2 we introduce a simple example with a relatively dynamic structure and limited quantities of data. Section 3 is used to describe the multiple levels of abstraction. Then, in Section 4, our current insights into modelling multiple levels of abstraction are summarized. Our solution to the problem is described in Section 5 with the Leeds Product Data Editor. Finally some conclusions and possibilities for further research are presented.

2. An example: Product family modelling

The change from the sellers' market of the fifties to the buyers' market we know today has resulted in a dramatic increase of product variants. Where products were originally offered in only a few variants, now some products can be produced in millions of (slightly) different variants. Examples can be found in car, aeroplane and medical equipment companies. In these situations, it is not economically viable anymore to make all variants to stock.

It is difficult to store all possible variants separately in a database because of the data redundancy caused by the commonality between the variants. Further, it is difficult to have insight into the family of variants as the relationships between the variants (i.e. the structure of the family) are not stored in the database and thus lost for the user and everyone else. Therefore, both a new family description and a new logistic concept is needed:

- product families should be described as product families, instead of a collection of individual product variants; and
- customers should be able to specify a variant of a product family, and the manufacturer should be able to assemble that specific variant to customer order.

The customer specification, together with a product family description, is used to generate all

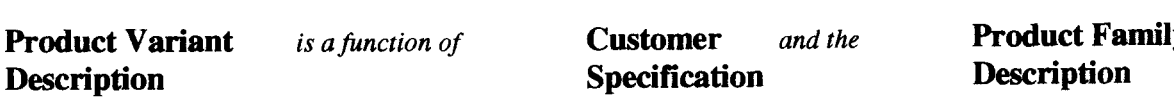


Fig. 1. Generating product information.

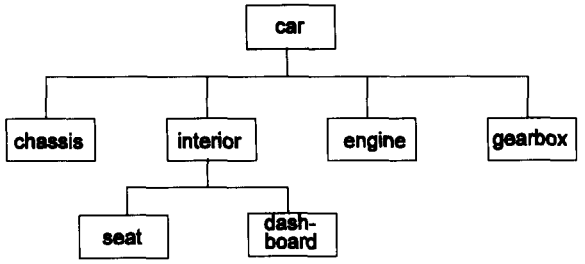


Fig. 2. Product family structure.

specific (manufacturing) documentation for that specification (Fig. 1) [2].

In order to enable the generation of a product variant description, specifically for a given customer specification, a product family description must be available. The creation of such a family description is normally the responsibility of marketing, engineering and design. It is an example of data where structural data is intertwined with repetitive data and shared between different users.

Example

Consider, for example, a car family. A car is, amongst other things, assembled from a chassis, an engine, a gearbox and an interior. The interior is assembled from the seats and the dashboard (see Fig. 2). The relationships between the different parts of the car are structural data, while the parts themselves are repetitive data which may itself be structured.

Further, a number of views can be determined, each having its specific requirements in the organization. The design process of such a car involves two different types of activity and produces data on different abstraction levels, namely:

- "structural data", shared by all cars of that family which defines the structure of a car;
- "repetitive data", which details the components of a car.

These different abstraction levels will be elucidated in the next section. First we will continue with this example.

Each of the aforementioned sub-assemblies can be considered as a product family. The interior, for example, is a product family which is assembled from the seats and the dashboard. All product families have a number of variants. The variety of the car family originates from the variety of its sub-assemblies. In turn, the variety of the interior originates from its sub-assemblies, i.e. of the seats and the dashboard.

The data structures behind today's engineering databases [3,4] typically necessitate the definition of every variant of a product family. This means the need for the separate description of possibly millions of variants of a car, where each description has a large commonality with descriptions of other variants. This redundancy of data is normally not acceptable.

When variants of a product family are physically assembled from component variants, it is possible to build the descriptions of the variants. The description of a car variant can be built from:

- the descriptions of the chassis, engine, gearbox and interior variants; plus
- the description of the car family (e.g. how do the component families fit together).

For example, the description of an interior variant can be built from the descriptions of the seat and dashboard variants, plus some information about interiors in general (e.g. how seat and dashboard families are normally assembled). Fig. 3 shows the built descriptions in black and the basic descriptions in white. Basic descriptions belong to basic, or so-called primary variants, while built descriptions belong to so-called assembly variants.

For the purpose of this paper, we will assume that all assemblies are assembled to customer order. The generation of product descriptions for

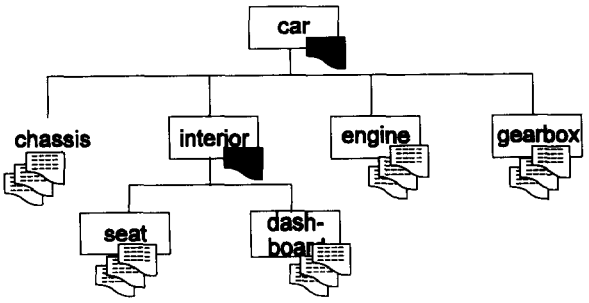


Fig. 3. Building documentation of assembled variants.

Table 1
Parameters and parameter values for the engine

Parameters	Parameter values
Engine size	1.8 litre, 2.0 litre
Turbo	yes, no

variants is in line with the nature of assemble-to-order manufacturing. Both physical stocks and data redundancy are reduced.

In repetitive situations, we do not want to build the descriptions of the individual product variants by hand. Preferably we want to generate all of the necessary information automatically from the customer order, as shown in Fig. 1. Therefore we need to introduce a defining mechanism where customers can describe a product variant in commercial terms and manufacturing can describe a product variant in terms of the product architecture and the primary variants from which the other variants are composed. This mechanism is called a generative bill-of-material [5].

The commercial terms in which a customer specifies a product variant are called parameters. Each parameter has a number of possible values from which the customer must choose to define a product variant. We will not elaborate on this subject, but the parameters that are relevant to specify the engine are listed in Table 1, as we will use these parameters in Section 4.

3. Multiple levels of abstraction

In the previous section, we indicated that there are two different aspects to the design of product families, namely, defining the structure of the product family and detailing the component variants that can occur in this structure.

An engineer who is responsible for designing cars in general will use a database to facilitate his or her design activities. These activities have, for a large part, a project management character. The information structure that (s)he needs for designing cars will be implemented in the database schema and will contain entities like:

- who is responsible for a product family;
- when its detailed design should start and finish;
- the relationships which exist with other product families; and
- where the variants of a certain product family will be manufactured.

A specific car family will be stored in a database, using such a database schema. When the data dictionary is level 1 in the list of abstraction levels, then the database schema for the project management of car development will be level 2, and the individual families will be found on the third level.

Other people are responsible for defining the possible variation in the product families which constitute the car rather than the architecture of the car itself: for example, the gearbox, the fuel injector and the seats. The possible variation of a product family is defined by a list of family-specific parameters. For example, an engine may be defined by its size and whether or not it is turbo-charged whereas a seat may be defined in terms of its material, degree of padding and type of headrest. For these people the parameters of each product family should be captured in a database schema so that variants can be defined in a database which conforms to this schema.

Thus, in total, four levels of abstraction can be determined in our example:

(1) the data-dictionary level is a data structure;
(2) the project management level supports the description of product families;
(3) product families are defined in terms of the project management data structures and define the sets of parameters in terms of their variants which must be defined;
(4) product variants are defined in terms of their family parameters.

These four levels are summarized in Fig. 4. It can be seen that levels 1 and 2 are data structures and that level 4 is instance data. However, level 3 is both a data structure (where the form of variants is defined) and an instance (of the project structure for a car family).

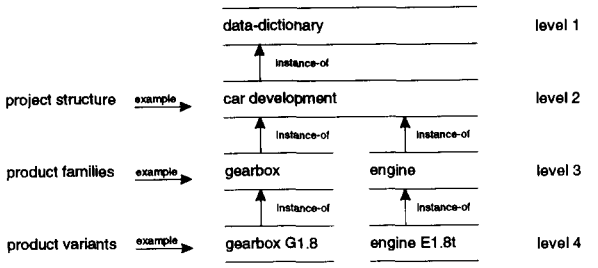


Fig. 4. Examples of classification.

Limitations of conventional database systems

Redundancy of data is introduced when these abstraction levels are implemented in separate three-layer databases:

The first type of redundancy concerns the different abstraction levels. It is not possible to model four levels of abstraction without repeating structural data in one database as repetitive data in another database. Conventional relational or object-oriented databases have only three levels, namely: data dictionary, tables/classes and tuples/instances.

The second type of redundancy concerns redundancy within a single level. Database tables (for example for variants of the engine and the gearbox) will have a number of similar attributes, which are repeated for tables. An example is given in Section 4.1.

The next section describes how today's techniques deal with the modelling of abstraction levels and data redundancy in general. It is shown that even modern object-oriented databases are not powerful enough to model this problem conceptually right. More advanced, but still academic, approaches offer better solutions.

4. Current practices and insights

In this section, we will describe three different approaches before we elaborate a fourth approach, which is based on the Leeds Product Data Editor, in Section 5. Firstly, we will discuss an implementation in a relational database. Secondly, we will describe the advantages and shortcomings of object-oriented databases. Finally, we will show a fundamentally different approach (3DIS) which has been proposed by Afsarmanesh and McLeod [1].

4.1. Relational approach

Relational databases make a clear distinction between data and structure [6]. Normally, there is a large volume of data and only a relatively small and stable structure. This structure is often called the database schema and is used by application programmes to work on the data. According to our previous figure, we will need a number of different, however related databases:

- "car development" as a project database for "specific product families" (see Table 2);
- several databases for product families, e.g. "engine" and "gearbox" (see Tables 3 and 4).

From this example, it can be seen that "engine" and "gearbox" appear both on the data level (Table 2), and on the schema level (Tables 3 and 4).

In this way data redundancy over different levels of abstraction is introduced. Further, the engine (gearbox) information that is presented in Table 2 is valid for all design variants of that engine (gearbox) which can be found in Table 3

Table 2
Attributes and data for "car development"

Product family name	Responsibility	Family start date	Family finish date	Main product relationships
Car	P. Breuls	1992-10	1995-2	Engine, Gearbox
Engine	H. Hegge	1993-3	1994-8	Ignition, Gearbox
Gearbox	E. Platier	1993-3	1994-6	Interior, Engine
Interior	R. Stekel	1993-7	1994-11	Dashboard, Seats
Chassis	T. Renkema	1992-10	1993-12	Interior, Suspension

Table 3
Attributes and data for the product family "engine"

Design variant	Fuel economy	Gearbox interface	Variant start date	Commercial parameters
E1.8	16 km/l	G1.8	1993-3	engine size = 1.8 and turbo = no
E2.0	18 km/l	G2.0	1993-4	engine size = 2.0 and turbo = no
E1.8t	15 km/l	G1.8	1993-8	engine size = 1.8 and turbo = yes
E2.0t	17 km/l	G2.0	1993-9	engine size = 2.0 and turbo = yes

Table 4
Attributes and data for the product family "gearbox"

Design variant	Gear ratio	Propeller shaft interface	Variant start date	Commercial parameters
G1.8	1:0.5678	S-manual	1993-4	engine size = 1.8 and automatic = no
G1.8a	1:0.6201	S-automatic	1993-4	engine size = 1.8 and automatic = yes
G2.0	1:0.7675	S-manual	1993-10	engine size = 2.0 and automatic = no
G2.0a	1:0.8945	S-automatic	1993-10	engine size = 2.0 and automatic = yes

(Table 4). It is however not possible to express the link between the engine (gearbox) tuple of Table 2 and the database schema of Table 3 (Table 4).

This complexity is further increased due to redundancy of structural information within a single level of abstraction. The attributes "variant start date" and "commercial parameters", for example, are needed by both the engine and the gearbox table (on level 3), and are repeated for both. However, not only the attributes are repeated, also the application programmes working on these attributes either:

- are duplicated and adapted for the different tables in which these attributes occur; or
- know about both database schema (the engine and the gearbox schema).

Both situations make an adaptation of the database schema more difficult and costly. The next section shows how object-orientation uses inheritance to remove this redundancy within a single level of abstraction. The redundancy over different levels of abstraction, however, continues to exist.

4.2. Object-oriented approach

Although object-oriented techniques [7] offer some powerful concepts for data modelling, there is still a general consensus that there is a strict borderline between structural data (classes) and specific data (objects). Instantiation is used to link the class and object levels and can be regarded as the most basic object-oriented reusability technique as objects reuse the structure which is defined at their classes. Classes can be regarded as instances of a meta-class, which is similar to the data dictionary of relational databases. With respect to different abstraction levels, though, the data redundancy is *not* solved.

In Section 4.1, we discussed the problem of redundancy at the structural level. Attributes need to be repeated as was shown in Table 3, where some attributes (design variant, variant start date and commercial parameters) are not unique for an engine and therefore repeated for the gearbox in Table 4. An object-oriented technique to remove this data redundancy is the use of structural inheritance. Attributes and application programmes that are shared by several classes are only defined once and then inherited to these classes. This mechanism is also known as specialization or the superclass/subclass relationship [7]. Fig. 5 shows how the attributes "design variant", "variant start date" and "commercial parameters" are inherited from a superclass "family". These attributes only need to be defined once, and can then be made use of by any subclass of "family".

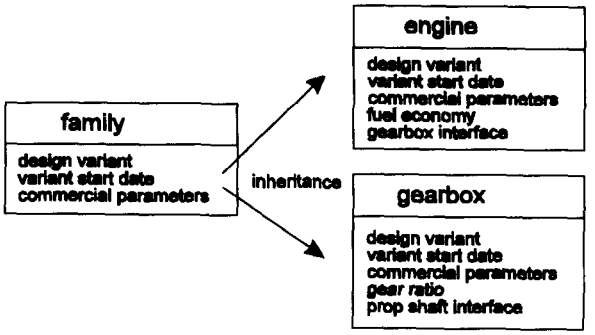


Fig. 5. Inheritance of structural information.

4.3. 3DIS approach

Afsarmanesh and McLeod [1] propose a database where all information including the data, the descriptions and classifications of data (meta-data), abstractions, operations, and constraints are treated uniformly as objects. Within this framework, 3DIS incorporates several predefined abstraction primitives, including a generalization hierarchy.

Objects and mappings are the two basic modelling constructs in 3DIS. Relationships among objects are modelled by "(domain-object, mapping-object, range-object)" triples. The structure of these triples is very much like the binary relationships in the binary semantic model [8]. Below some mappings are given, which are used for the car example in Figs. 6–8:

Member/type mappings
- instantiation: Has-member: type objects → P(atomic/composite objects)
- classification: (Is-a-member-of)

Member-mapping/type mappings
- decomposition: Has-member-mapping: type objects → P(member-mappings)
- aggregation: (Is-a-member-mapping-of)

```
( Product, Has-member-mapping, {
                    Has-product-family-name,
                    Has-responsibility,
                    Has-family-start-date,
                    Has-family-finish-date,
                    Has-main-product-relationships } )
```

Fig. 6. Car development level.

```
( Engine, Has-member-mapping, {  Has-design-variant,
                                 Has-fuel-economy,
                                 Has-gearbox-interface,
                                 Has-variant-start-date,
                                 Has-commercial-parameters } )

( Engine, Is-a-member-of, { Product } )

( Commercial-parameters, Has-member-mapping, { Has-name,
                                               Has-value } )

( Engine, Has-related-product, { Ignition } )

( Engine, Has-related-product, { Gearbox } )

( Engine, Has-member, {  E1.8,
                         E2.0,
                         E1.8t,
                         E2.0t } )
```

Fig. 7. Engine definition level.

Every identifiable information fact in an application environment corresponds to an object in a 3DIS database. Simple, compound, and behavioural entities in an application environment, attributes of objects and relationships among objects, as well as object groupings and classifications are all modelled as objects. What distinguishes different kind of objects in a 3DIS database is the set of structural and nonstructural (data) relationships defined on them.

The concept which we have exploited to model multiple levels of abstraction is "instantiation/classification". However the semantics of these terms are not defined in Ref. [1]. Our interpretation is that an instance of a class is an object whose type is the class and which has values for the attributes of that class. For example, *Engine (1.8 litre, with turbo)* is an instance of the object engine defined as *Engine HAS (size (1.8 litre OR 2.0 litre) AND turbo (with OR without))*.

```
( E1.8, Is-a-member-of, { Engine } )

( E1.8, Has-fuel-economy, { 16 km/litre } )

( E1.8, Has-gearbox-interface, { G1.8 } )

( E1.8, Has-variant-start-date, { 1993-3 } )

( E1.8, Has-commercial-parameters, { Engine-size=1.8, Turbo=no } )
```

Fig. 8. Engine instance level (E1.8 only).

Of course this interpretation could be incorrect. Another problem with 3DIS is that it appears to be weakly typed; it is not clear how (or even if) the conformance of instances to their types is guaranteed. In practice the conformance of instances to their types is essential. For example, it must not be possible to define an engine without giving exactly two values of the correct types: otherwise the engine definition would be incomplete or incorrect in some other way.

Finally, 3DIS does not appear to support modularization. The effect of this is that the data structures which need to be populated during different design activities are mixed with instance data and are therefore not readily visible. Understandability is an important requirement of data models [9] and a 3DIS database of a real engineering example would be unlikely to support this requirement.

So, whilst 3DIS supports the implementation of multiple levels of abstraction, a better way of describing the data and its structure is required. The Leeds Structure Editor, described in the next section, provides such a capability. Once the structure and content of the data have been determined using such a tool it could be implemented, with the appropriate user interface and type checking tools, in a database system such as 3DIS.

5. Applying the Leeds Structure Editor for product family modelling

5.1. An introduction to the Leeds Structure Editor

The Leeds Structure Editor (SE) is a tool that has been developed to create and edit product data structures. One of the problems of today's computer systems is that people must restructure the contents of their minds so that software applications can use the information held there. The SE has been conceived with the idea that people, particularly engineers, can interact with the computer in a way which gives them contact with their structured data [10]. Several pieces of software have been written that make use of data from the Leeds Product Data Editor, including geometric evaluators, vector packages and cutter path generation programmes.

SE's data dictionary has a number of constructs which allow engineers to create directed graphs, consisting of nodes and relations. These are:

- collections (COL) of other nodes (group);
- selections (SEL) of other nodes (choice);
- lists of nodes, which are all of the same type;
- atoms, which are leaves in the hierarchical structure, the basic types of the SE.

The hierarchical structure that is created with these constructs is called the meta-structure, comparable with a schema in a database (level 2 in Fig. 4). The data dictionary (level 1 in Fig. 4) is called the bootstrap meta-structure and is hardwired in the SE. All meta-structures can be regarded as instances of this bootstrap meta-structure. The meta-structures can be populated (instantiated) with values in a similar way as in databases. Fig. 9 shows how the car structure of Fig. 3 (completed with variants) could be modelled in the SE. Here, the car family is modelled as a meta-structure, while the variants are modelled as instances of this meta-structure.

The above meta-structure has the main disadvantage that it can only be used for modelling variants of the car product family. It is not generic enough for modelling completely different families, e.g. aeroplanes and medical equipment. This deficiency can, however, be overcome by a characteristic of the SE, namely an implementation of λ-calculus with which "multiple abstraction levels" can be captured.

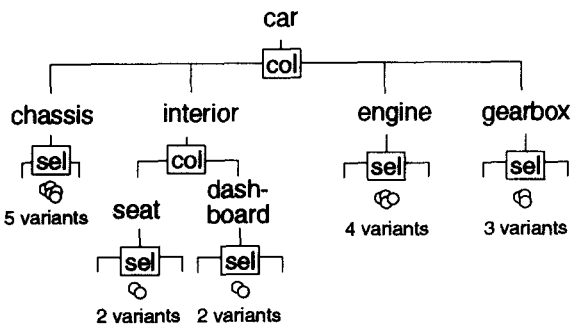


Fig. 9. Car modelled in the Leeds Structure Editor.

5.2. Using λ-calculus for modelling multiple abstraction levels

The solution adopted in the structure editor is to model the form of final instances, i.e. the lowest abstraction level, in the meta-structure, and to model the intermediate abstraction levels with λ-calculus. This means for our purposes that:

- the structure and content of a product variant are modelled in the meta-structure (level 2 in Fig. 4);
- specific variants for customer orders are modelled as instances of the meta-structure (level 4 in Fig. 4);
- product families as cars, trucks and aeroplanes are modelled using λ-calculus (level 3 in Fig. 4).

Fig. 10 shows the meta-structure for variants. This meta-structure is defined recursively because components of a variant can be variants again. The LIST construct is used to model an arbitrary number of components, which are all variants again.

Fig. 11 shows a small part of a car assembly variant (coded AV11) and its engine (coded E1.8t). This variant is an instance of the meta-structure of Fig. 10. Now, we will use this specific instance to model the car family with its component families. We will replace specific instances by a parametrized λ-function. The parameters of this function are used to define the precise primary variants of a family. An example will clarify this.

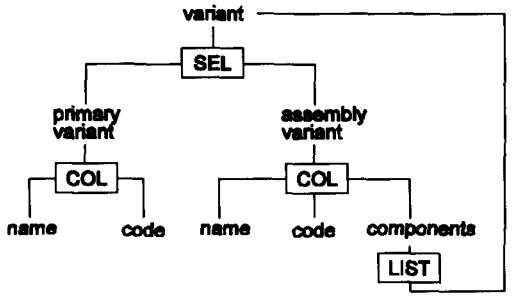


Fig. 10. Meta-structure for variants.

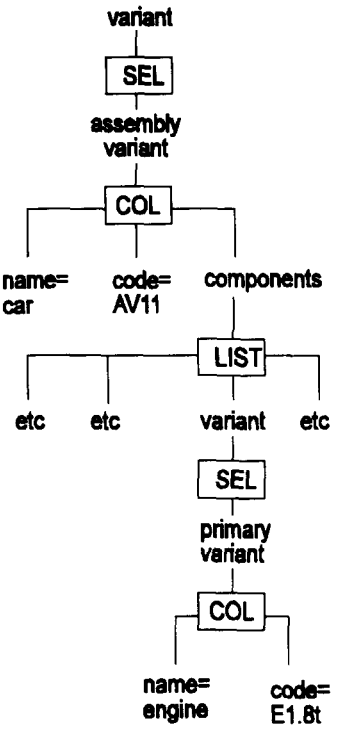


Fig. 11. A specific car variant.

First we will look at a component family of the car, namely the engine. The engine has four variants, of which the selection is dependent on the commercial parameters "engine size" and "turbo". If we model the engine as a function, the abovementioned commercial parameters will be used to invoke this function.

Engine(engine size = [1.8, 2.0],
turbo = [yes, no])

with body:

IF engine size = 1.8 AND turbo = no
THEN E1.8
IF engine size = 2.0 AND turbo = no
THEN E2.0
IF engine size = 1.8 AND turbo = yes
THEN E1.8t
IF engine size = 2.0 AND turbo = yes
THEN E2.0t

The IF–THEN statements in the body of the function are used to determine the right primary variant of the engine. One of these variants will be returned by the function, when the function is invoked with parameter values.

We now can replace a specific variant of the engine by the function Engine:

code = E1.8t

is replaced by

code = Engine (engine size = [1.8, 2.0],
turbo = [yes, no])

When we give values to the parameters of this function and evaluate the function, then the function structure is removed and the previous value of the code of the engine is replaced by the result of the function Engine:

code = Engine (engine size = 2.0,
turbo = no)

is evaluated and results in

code = E2.0

which is an instance of the meta-structure of Fig. 10 again.

This approach might look rather complex, but has proved to be satisfactory for modelling a product family. The effort of parametrizing variants to create families is small. Further, the different models (on different abstraction levels) are small and easy to see. The λ-functions which are used for modelling component families can easily be shared by different parent families, thereby eliminating data redundancy and stimulating reuse.

5.3. Summary of approaches

Table 5 gives an overview of the relational, object-oriented, 3DIS and Leeds Structure Editor approaches.

From the table it can be seen that both the relational and the object-oriented approaches can offer three levels of abstraction to model four levels of abstraction; thus two three-layer models are needed to model four levels, which results in duplication (indicated by italics). Product families (level 3) are instances of project management (level 2) and are repeated as tables or classes (level 2) of the product variants (level 3). This results in a relationship from product families as instances to product families as classes, which is difficult to express in relational and object-oriented databases. The benefit of object-oriented databases over relational databases is the possibility to inherit structural information, which reduces the redundancy of information on an abstraction level.

Both the 3DIS and the SE approaches support the direct modelling of multiple levels of abstraction. The SE is geared towards allowing people to see and interact with the structure and content of their data and a general-purpose graphical user

Table 5
Summary of approaches

Level	Relational		Object-oriented		3DIS	Leeds SE
1	Data dictionary		Meta-class		Object	Bootstrap meta-structure
2	Project management	*Product families*	Project management	*Product families*	Project management	Meta-structure for variants
3	*Product families*	Product variants	*Product families*	Product variants	Product families	Product families (λ-calculus)
4					Product variants	Product variants

interface supports this interaction. 3DIS, on the other hand, is a database system which is best used in conjunction with sophisticated, and possibly data structure specific, user and application interfaces. In practice both systems have a role to play: the SE allows people to prototype and test their data structures whereas 3DIS provides a database environment in which the tested data structures can be implemented and used in a production environment.

6. Conclusions

This paper has tried to raise an awareness with respect to abstraction levels. The development of products, and product families in particular, shows that there are modelling problems which cannot be solved directly in relational or object-oriented databases. Forcing multiple abstraction levels in three database levels introduces more complex schema with inherent redundancy. Although this problem is generally known with experienced database designers, not much attention has been paid to this subject in literature or software design. Two academic approaches, the 3DIS and the Leeds Product Data Editor, have been developed to tackle multiple abstraction levels. They appear to be appropriate for modelling product families, but supported software of this kind is not yet available.

Acknowledgement

This work has been partly funded by the ACME Directorate of SERC Grant No. GR/H 24266.

References

[1] H. Afsarmanesh and D. McLeod, "The 3DIS: An extensible object-oriented information management environment", *ACM Trans. Inf. Syst.*, Vol. 7, No. 4, October 1989, pp. 339–377.

[2] E.A. van Veen, *Modelling Product Structures by Generic Bills-of-Materials*, Elsevier, Amsterdam, 1992.

[3] STEP Part 44, ISO-10303-44, Product Data Representation and Exchange, Integrated Generic Resources: Product Structure Configuration, Draft International Standard, 1993.

[4] P. Gu, " PML: Product modelling language", *Computers in Industry*, Vol. 18, 1992, pp. 265–277.

[5] F.J. Erens, H.M.H. Hegge, E.A. van Veen and J.C. Wortmann, "Generative bills-of-material: An overview", *Proc. IFIP'92*, Elsevier, Amsterdam, 1992.

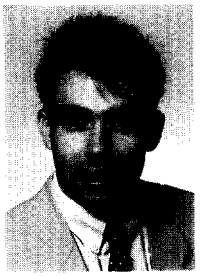
[6] J.J. van Griethuysen, "Concepts and terminology for the conceptual schema and the information base", ISO/TC97/SC5 N695, 1982.

[7] W. Kim (Ed.), *Object-Oriented Concepts, Databases and Applications*, ACM Press, 1989.

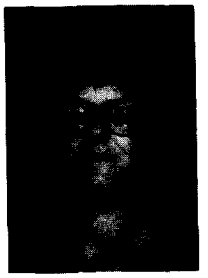
[8] J.R. Abrial, "Data semantics", in J.W. Klimbie and K.L. Koffeman (Eds.), *Data Management Systems*, North-Holland, Amsterdam, 1974.

[9] C. Batini, M. Lenzerini and S.B. Navathe, "A comparative analysis of methodologies for database schema integration", *ACM Comput. Surv.*, Vol. 18, No. 4, December 1986, pp. 323–364.

[10] A. McKay, "The Structure Editor approach to product description", University of Leeds, ISS Project Research Report, ISS-PDS-Report-4, June 1988.



Frederik Erens gained his degree in Information Science in 1990, after which he became a consultant at KPMG Lighthouse. Simultaneously, he started as a Research Engineer in the Department of Industrial Engineering and Management Sciences, Eindhoven University of Technology. His current research interests include product modelling, integral logistics and the development of complex products which are offered in a large variety.



Alison McKay gained her degree in Mechanical Engineering in 1982 and then spent two years in industry before becoming a Research Engineer in the Department of Mechanical Engineering. Her research interests include the representation and integration of product data. Since 1988 she has taken an active role in STEP, particularly in integration, development methods and as a editor of Part 41. Currently she is leading the product data modelling and integration activities of the SERC/ACME-funded MOSES (Model Oriented Simultaneous Engineering Systems) project.

Susan Bloor gained her degree in Mathematics in 1962 and her PhD in 1965. She is Senior Computational Advisor in the Department of Mechanical Engineering. Her research interest is in engineering product data modelling and its role in systems integration. She has played a leading role in the Department's research activities in this field for many years through a continued involvement in the industrially sponsored Geometric Modelling Project, as Principal Investigator on Teaching Company Schemes and as Project Manager for various European-funded projects.