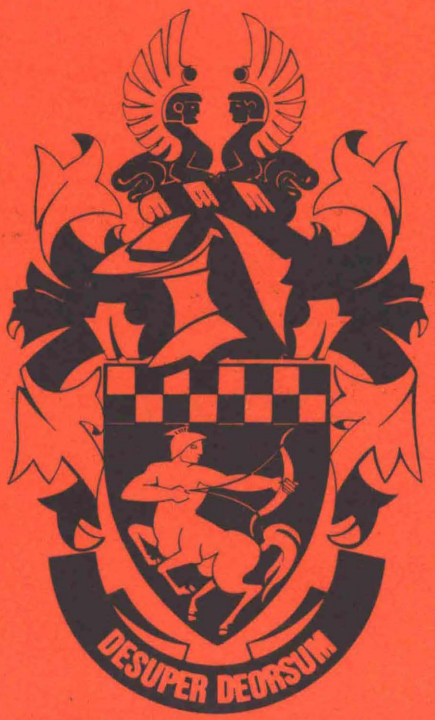
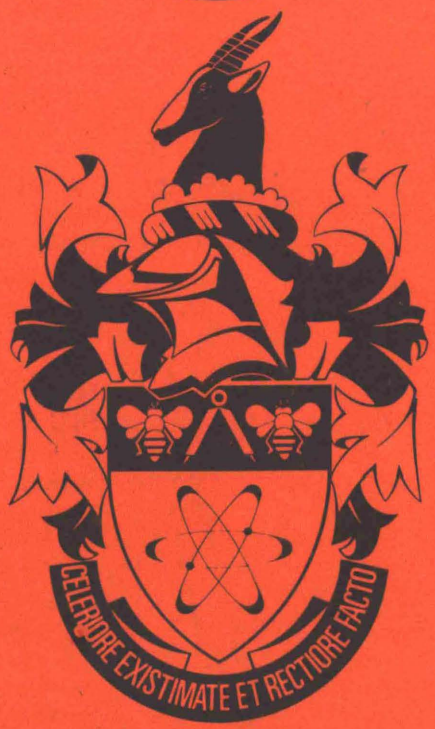




Quaestiones Informaticae

Vol. 2 No. 3 September, 1983

Quaestiones Informaticae

An official publication of the Computer Society of South Africa and of the South African Institute of Computer Scientists

'n Amptelike tydskrif van die Rekenaarvereniging van Suid-Afrika en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

Editor: Prof G. Wiechers
Department of Computer Science and Information Systems
University of South Africa
P.O. Box 392, Pretoria 0001

Subscriptions

Annual subscriptions are as follows:

	SA	US	UK
Individuals	R6	$7	£3,0
Institutions	R12	$14	£6,0

Circulation manager
Mr E Anderssen
Department of Computer Science
Rand Afrikaanse Universiteit
P O Box 524
Johannesburg 2000
Tel.: (011) 726-5000

Quaestiones Informaticae is prepared for publication by Thomson Publications South Africa (Pty) Ltd for the Computer Society of South Africa and the South African Institute of Computer Scientists.

Add: The Automated Database Design Tool

S. Berman
Department of Computer Science, University of Cape Town, South Africa

Abstract
An automated database design tool called ADD has been developed at the University of Cape Town. This system obtains a requirements specification from a user and from this generates a relation scheme and a Codasyl schema. This is intended to be a prototype database. The specification is submitted using SDM, the Semantic Database Model. Semantic information in this model is included in the schema as integrity constraints.

1. Introduction

Database design is a difficult task which continues to present problems to theoreticians and practitioners alike. Currently no universally-accepted database design methodology exists, and the greatest part of database research is directed towards solving this problem. Some useful results have begun to emerge, such as normal form theory for relational database design. In the realm of network databases, there are three aids: data models, prototype databases and automated design tools. The ADD system is a database design tool which incorporates all these three facilities. The ADD design methodology is based on the SDM data model, and involves automated conversion of such a model to a prototype network database. After discussing the reasons for developing ADD, the SDM model is briefly described. Thereafter, the user interface, relation scheme design and network synthesis algorithms are outlined. In conclusion, some of the results of implementing this system are presented.

2. Motivation

The difficulties experienced in database design arise through inadequacy on the part of both users and designers. Frequently the user does not fully understand his data, cannot describe it accurately and completely, and cannot distinguish irrelevancies. Furthermore, there is generally a communication gap between user and database designer, and hence the latter can misinterpret the requirements. A data model facilitates recognising and understanding data relationships, and provides a fairly natural way of describing requirements in an unambiguous manner.

Even if the specifiation is correct and complete, database design is still a difficult task, because there are so many options facing the designer when deciding both logical and physical structure. Poor designs result if the structure is arrived at by examining processing requirements, rather than the fundamental properties of the data itself. The ADD system was designed to alleviate this by automatically generating a prototype database. Its logical structure should require minimal, if any, alteration when the final schema is decided. The synthesised network is based soley on a description of the data itself, and not on envisaged transactions. The system generates several listings showing the design decisions taken, which should assist the designer of the final operational database.

3. SDM — The Semantic Database Model

SDM is a data model developed by Hammer and McLeod, which first appeared in the literature in 1981 [1]. In such a model, the real world is described by means of classes and their attributes. A class represents an entity type; that is, an object of interest in the environment. Its attributes are the properties that such entities possess.

Every attribute has an associated valueclass. This specifies what type of value(s) it assumes, and can either be STRINGS, i.e. printable values, or some SDM class. For example, SHIPS attribute Type would have a valueclass STRINGS, e.g. 'merchant' or 'fishing', Captain would have valueclass OFFICERS, where OFFICERS is some class in the model. From this it can be seen that entities represent themselves, they are not represented by means of keys as in the relational model and others. This simplifies the model and makes it easier for non-DP persons to understand.

Relationships between entities can be represented either as classes, or as properties of the objects involved. For example, the relationship between a ship and an officer can be a class ASSIGNMENTS with attributes Ship-Assigned and Officer-Assigned. Alternatively, the class SHIP can have attribute Captain (valueclass OFFICERS) and/or the class OFFICERS can have attribute Ship-On (valueclass SHIPS). When a relationship is represented as a class, attributes of the relationship can be specified (example Assignment-Date).

The concept of generalisation [2] or subtyping is incorporated. A subclass of C defines a subtype or "special" type of C. As an example, TANKERS is a subclass of SHIPS. A subclass definition includes a condition which objects of the superclass must satisfy in order to belong to that subclass. Subclasses inherit all attributes of their superclass in addition to their own attributes. Subclasses of STRINGS, which is "system-defined", can be given to describe permissable values for attributes. This enables INTEGERS, DATES, PERSON-NAMES, etc to be distinguished from arbitrary character strings.

Grouping classes are used to describe classes comprising groups of like entities. CONVOYS would be a grouping of SHIPS. Each member of CONVOYS consists of not one but several SHIPS entities. The definition of a grouping class can stipulate how the groups are to be formed. Thus grouping class SHIPTYPE-GROUPS can be specified as a "Grouping of SHIPS on common value of Type". This causes all "fishing" vessels to constitute one grouping, etc. Classes which are neither subclasses nor groupings are called base. Each of these has one or more keys specified for it.

The semantic expressiveness of SDM can be seen when considering attribute descriptions. Each attribute must have a valueclass specified and must be designated either "member" or "class". A member attribute is applicable to individual entities in that class (e.g Type, Size, Hullnumber); the class attributes describe the class as a whole (e.g. Number-of-Ships, Average-Ships-Size). Where applicable, attributes can be described by properties such as "multivalued" (for repeating attributes), unchangeable (e.g. Birthdate) and cannot be null, amongst others.

Derived data can be included if their derivation can be explicitly defined in terms of other information in the model. Several attribute derivation primitives exist for this purpose. Some of these are: average, minimum sum, maximum, arithmetic expression, same as, subvalue, intersection, union and set difference. Subvalue enables a subset of a multivalued (repeating) attribute to be defined. For example, Last-Two-Inspections can be specified as subvalue of Ship-Inspections, where Order-for-Ship is less than 3. Intersection, union and set

difference can be applied to two multivalued attributes in a similar way.

SDM was chosen for the ADD system after studying a wide selection of existing data models. The main reason for this choice is that SDM is highly semantically expressive, providing many modelling tools, yet is sufficiently simple for a non-DP person to understand. In other models, a trade-of between expressiveness and simplicity is evident. Furthermore, SDM embodies only the meaning and nature of the data, and does not include any description of processing requirements. SDM could perhaps be criticised for not including functional or multivalued dependencies. This was not considered a disadvantage as these concepts are difficult for computer-naive people to understand and recognise in their environment. It was considered essential that the user should be able to specify the model himself.

4. The User Interface

The task of obtaining an SDM specification from a non-DP person is indeed an awesome one. The policy adopted was that of gradually introducing each SDM concept to the user in turn. Thus the novice is first asked for classes, then attributes. Subclasses and grouping classes are only introduced subsequently; and attribute derivations are considered in the final stages.

A menu-driven system was chosen as the safest approach, with large menus being split into several smaller ones. The top-level menu offers a choice between data entry, session termination, help, perusal and model editing. All data entry, i.e. of classes, attributes, etc, is handled in the following manner:

The user can submit a description of his organisation and its objects using natural language. However, only words in capital letters are recognised by the system. This is a slight modification of the data capture method used by Palme[3], with capitals instead of special symbols. They are easier to type, and a word that starts in lower case but ends in upper case is accepted. In this way a beginner can submit the following when asked for classes: "the univerSITY has DEPARTMENTS divided into courSES given by LECTURERS these are taken by STUDENTS for CREDITS" . . . The experienced user would simply give "UNIVERSITY DEPARTMENTS COURSES LECTURERS STUDENTS CREDITS".

When users have given all classes (or all attributes, etc), they respond "QUIT". They are then asked to supply details about the new items, so that a complete SDM model results. Since it can be dangerous to interrupt his train of thought[4], this process can be terminated prematurely by the user. It can be ommitted entirely, if he wishes, by submitting say "QUIT S", instead of "QUIT". This causes detail specification to be preempted and subclasses ("S") can immediately be entered instead. This mode of operation can be employed by experienced users to type-ahead, rather than be presented with the menu each time.

Perusal enables a user to review the current version of his model. This is helpful if he has forgotten his earlier work, or wishes to check that he is proceeding correctly. He can edit any item description by naming the item and then replacing errors by new values. An extremely simple method of editing is used, involving menu-selection. This is preferable to a special editing language, which is generally confusing to a novice.

Help can be requested at any stage by responding "?". This causes the program's most recent display to be expanded upon, giving greater detail on what is required. For this reason, all initial displays are brief, to avoid irritating the experienced user. If too many successive help requests are received, the user is referred to his User Manual.

When the model is completed, an optional component of the system may be executed. This allows functional dependencies to be specified and can be helpful if the user is capable of working with these, or is guided by the database designer.

5. Relation Scheme Design

The SDM model is converted into a relation scheme in the following manner: In general, all the attributes of a class constitute one relation describing that class. Exceptions occur in the following cases: Each candidate key for the class is removed to form a separate relation, and is replaced by an artificial key. Thus, for example, if Name is the key for OFFICERS, it is replaced by OFFICERS≠ (with domain INTEGERS), and another relation (OFFICERS≠, Name) is created. Not only can this save space, but it also ensures that the relation is in 2nd Normal Form, and that embedded dependencies between prime attributes are removed.

Multivalued (repeating) attributes also contribute separate relations, e.g. (SHIPS≠, Crew), so that the relation scheme is in 1st Normal Form, and data aggregates are replaced by their subfield components. A separate relation is used to store all class attributes, since these do not have multiplicity. If dependencies were specified in the model, the attributes involved form a new relation and the dependent ones are removed from the original. Similarly, any attribute derived from single-valued attribute(s) is also separated out. In this way, a model for which functional dependencies were specified is translated into a Boyce-Codd Normal Form scheme. Otherwise, a 2nd Normal Form schema results.

There are other types of relations in the relation scheme generated. An additional relation is created for each grouping class. As an example, relation (CONVOY≠,CONTENTS) enables several SHIPS to be stored "opposite" a CONVOY number. To handle subclasses, relations of the form C≠, CTYPE are generated, for all base classes C on which subclasses were defined. In this way a TANKER (subclass of SHIPS) would have its key in at least three relations: that for SHIPS, that for TANKERS and that giving (SHIPS≠,SHIPSTYPE). This is more efficient than duplicating SHIPS attributes in the TANKERS relation, and makes the relational database easier to use.

6. Network Schema Synthesis

The construction of a DMS 1100 Codasyl schema[5] is implemented as a 3-phase process. The first translates the relation scheme into a network structure; the next uses the SDM model to refine this and introduce integrity constraints. In the final phase, physical parameters are chosen and a complete schema created.

The first phase converts each relation in tern into records, sets and items. A record type is formed for each class, comprising the attributes in the relation describing that class. Repeating attributes constitutes a separate record type, connected to the class they characterise as member of a set. A functional dependency also forms a new record type, and is made the owner of a set linking it to the class it describes.

Relationships translate into sets, with their one:one/one:many/many:many nature distinguishing the owner from the member. A confluent hierarchy is used for the many:many case. The relationships in the network are derived from those attributes whose valueclass (domain) is not STRINGS, but indicates another class in the model. Thus throughout the network generation module, before an attribute is placed in a record, its domain is inspected. If this implies a class, a set is created instead of a field. (A class has "classname" as column name and "integer" as domain; while an attribute is represented by "attributename" and valueclass, respectively.) Thus when handling attributes, where there is no "≠" in the column name, the first step is to examine the domain for a "≠". If none exists, the attribute becomes an item in some record. If one is found, the "≠" is removed and what remains identifies the valueclass of the attribute. In such a case set(s) are required. For example, if "CAPTAIN, OFFICERS≠" is encountered in the SHIPS relation, then a set OFFICER→SHIPS is created. "Loops" are detected and replaced by two sets, in the manner advocated by Date[6]. This is necessary because the same record cannot be both owner and member record type in a DMS 1100 set.

Grouping classes are generally implemented as a set with the grouping class as owner and grouped class as member (example SHIPTYPE-GROUPS→SHIPS). Thus each such set occurrence represents one group. Subclasses are implemented in the following way.

If TANKERS is a subclass of SHIPS, then in addition to their own attributes, TANKERS have all the attributes of ordinary SHIPS. As most SHIPS will not be TANKERS, it is inefficient to declare a SHIPS record to contain both SHIP and TANKER attributes, as the latter will be null in most instances. It is also dangerous to treat SHIP and TANKERS as completely separate entities, because then if user asks for all SHIPS in the database, he must know to list all TANKERS too; also, all relationships in which SHIPS participate would have to be duplicated for the TANKERS record type. Thus separate record types are used, but they are connected by a set, such as SHIPS->TANKERS.

By adding flags to the superclass record, the following can be verified by means of appropriate CHECK and RESULT clauses:

1. A TANKER can only exist in the database if it is linked to 1 and only 1 SHIPS occurrence.
2. A SHIPS record cannot have more than on TANKER record linked to it.
3. A TANKER can only be linked to a SHIP if that SHIP meets the criteria set down in the TANKERS definition.

Verifying the latter, as well as attribute derivations, provided the greatest complexity in ADD. The problem lies essentially in the fact that in SDM one can specify relationships between two semantically distant attributes. This is beyond the capabilities of the DMS 1100 integrity facilities, since CHECK and RESULT clauses can only handle items in the same record or set.

In ADD, integrity constraints are incorporated wherever possible, to verify subclasses and derived attributes. Certain derivations are also examined in order to simplify the database structure. As an example, if attribute A is the union of B and C, this can mean replacing six sets by two.

In the final phase of schema generation the simplest physical structure is chosen for the database. This should be adequate for the prototype, but would have to be altered when the final schema is chosen. The only complexity that can arise occurs with multi-level paths in a SET SELECTION clause. ADD determines whether these are necessary, and chooses USING and ALIAS items along the path.

7. Conclusion

The ADD system automatically produces a schema which can be used as a prototype database. Its strength lies in its choice of a logical structure, which should require minimal, if any, alterations. The wealth of information in the SDM model is used to define records, sets and items in the best possible way. The characteristics of attributes, interclass connections and attribute derivations which enhance one's understanding of the organisation, are utilised to improve the network. As the logical structure is based solely on a description of the data, and not on functional requirements, it should not need to be re-organised when processing requirements change. Integrity constraints are included where permitted by DMS 1100. Add also relieves the human designer of much of the tedium associated with the specification of physical criteria. For example, sort keys are designated where appropriate, ALIAS items are declared and singular sets chosen as VIA sets wherever these exist.

The ADD system has shown clearly that considerable advantages exist when a data model forms the basis of design. Its implementation has high-lighted the fact that a data model should not provide features beyond the scope of conventional Database management Systems. A modified version of SDM has accordingly been designed.

The other major contribution of ADD lies in the fact that it enables a computer user to create an SDM model. Experimentation with novices showed that after some inital difficulty, a user can indeed construct a model with little, if any, assistance. Problems encountered in the beginning arise largely through misinterpretation of ADD terminology. Hence if this is discussed with the database designer before commencing, a user of average inteligence should be able to create his model relatively easily and quickly.

Thus ADD has shown that it is possible to develop a database design methodology, which is partially automated and directly involves the user in requirements specification. The entire design process cannot be automated; the analytical powers, intuition and creativity of the human mind are essential. All that can be done is to automate as much of the design process as is reasonable, to alleviate the task of human(s) involved. The ADD system is a step in this direction.

References

[1.] Hammer, M and McLeod, D. "Database Description with SDM; a Semantic Database Model". ACM Transactions on Database Systems, Vol. 6, No. 3 1981, pp 351-386.

[2.] Smith, J M and Smith, D C P. "Database Abstraction: Aggregation and Generalization". ACM Transactions on Database Systems, Vol. 20, No. 6, 1977, pp 105-133.

[3.] Palme, J. "A Human-Computer Interface for Non-Computer Specialists". Software Prac. and Exper., Vol. 9, 1979, pp 9-19.

[4.] Martin, J. "Design of Man-Computer Dialogue". Prentice Hall, Englewood Cliffs, NJ, 1973.

[5.] Sperry Univac. "Data Management System (DMS 1100)". UP7909, Rev 3A, St Paul, Minn., 1972.

[6.] Date, C J. "An Introduction to Database Systems". 3rd Ed., Addison Wesley, 1981.

Notes for Contributors

The purpose of this Journal will be to publish original papers in any field of computing. Papers submitted may be research articles, review articles, exploratory articles of general interest to readers of the Journal. The preferred languages of the Journal will be the congress languages of IFIP although papers in other languages will not be precluded.

Manuscripts should be submitted in triplicate to: Prof. G. Wiechers at:

Department of Computer Science
University of South Africa
P.O. Box 392
Pretoria 0001
South Africa

Form of manuscript

Manuscripts should be in double-space typing on one side only of sheets of A4 size with wide margins. The original ribbon copy of the typed manuscript should be submitted. Authors should write concisely.

The first page should include the article title (which should be brief), the author's name, and the affiliation and address. Each paper must be accompanied by a summary of less than 200 words which will be printed immediately below the title at the beginning of the paper, together with an appropriate key word list and a list of relevant Computing Review categories.

Tables and figures

Illustrations and tables should not be included in the text, although the author should indicate the desired location of each in the printed text. Tables should be typed on separate sheets and should be numbered consecutively and titled.

Illustrations should also be supplied on separate sheets, and each should be clearly identified on the back in pencil with the Author's name and figure number. Original line drawings (not photoprints) should be submitted and should include all relevant details. Drawings, etc., should be submitted and should include all relevant details. Drawings, etc., should be about twice the final size required and lettering must be clear and "open" and sufficiently large to permit the necessary reduction of size in block-making.

Where photographs are submitted, glossy bromide prints are required. If words or numbers are to appear on a photograph, two prints should be sent, the lettering being clearly indicated on one print only. Computer programs or output should be given on clear original printouts and preferably not on lined paper so that they can be reproduced photographically.

Figure legends should be typed on a separate sheet and placed at the end of the manuscript.

Symbols

Mathematical and other symbols may be either handwritten or typewritten. Greek letters and unusual symbols should be identified in the margin. Distinction should be made between capital and lower case letters between the letter O and zero; between the letter l, the number one and prime; between K and kappa.

References

References should be listed at the end of the manuscript in alphabetical order of author's name, and cited in the text by number in square brackets. Journal references should be arranged thus:

1. ASHCROFT, E. and MANNA, Z. (1972). The Translation of 'GOTO' Programs to 'WHILE' Programs, in *Proceedings of IFIP Congress 71*, North-Holland, Amsterdam, 250-255.
2. BÖHM, C. and JACOPINI, G. (1966). Flow Diagrams, Turing Machines and Languages with only Two Formation Rules, *Comm. ACM*, **9**, 366-371.
3. GINSBURG, S. (1966). *Mathematical Theory of context-free Languages*, McGraw Hill, New York.

Proofs and reprints

Galley proofs will be sent to the author to ensure that the papers have been correctly set up in type and not for the addition of new material or amendment of texts. Excessive alterations may have to be disallowed or the cost charged against the author. Corrected galley proofs, together with the original typescript, must be returned to the editor within three days to minimize the risk of the author's contribution having to be held over to a later issue.

Fifty reprints of each article will be supplied free of charge. Additional copies may be purchased on a reprint order form which will accompany the proofs.

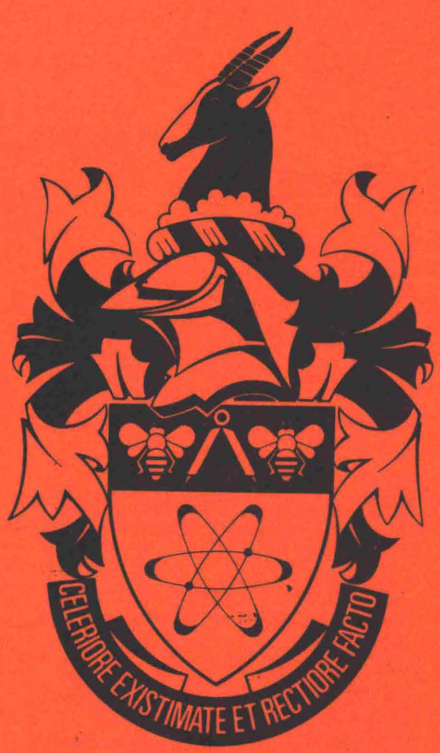
Only original papers will be accepted, and copyright in published papers will be vested in the publisher.

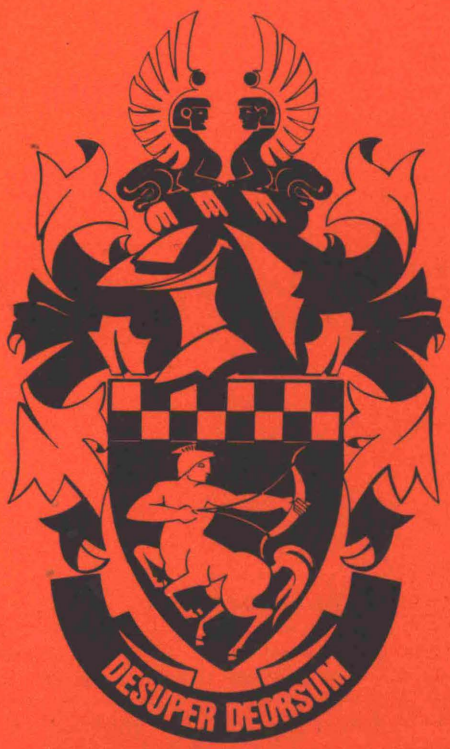
Letters

A section of "Letters to the Editor" (each limited to about 500 words) will provide a forum for discussion of recent problems.

Hierdie notas is ook in Afrikaans verkrygbaar.

Quaestiones Informaticae





Contents/Inhoud

*Presented at the second South African Computer Symposium held on 28th and 29th October, 1981.

MILITHO