4th SUID-AFRIKAANSE REKENAARSIMPOSIUM
SOUTH AFRICAN COMPUTER SYMPOSIUM
HOLIDAY INN PRETORIA
JULIE 1 – 3 JULY 1987
Proceedings

of the

4th South African Computer Symposium

Holiday Inn, Pretoria

1 – 3 July 1987

edited by

Pieter Kritsinger

Computer Science Department
University of Cape Town
PREFAE

Computer science is an emerging discipline which is having difficulty in being recognised as a worthy member of the sciences. I will paraphrase John Hopcroft, co-winner of the 1986 Turing Award, when, during a recent interview, he said that the primary reason for the lack of recognition, is the age of our researchers. Probably not one of the researchers who presented their work at this symposium is older than 45. I know of no computer scientist in South Africa who is in a position where (s)he can affect funding priorities. As far as I know we have no representation on any of the committees of the Foundation for Research Development and for our Afrikaans speaking fraternity, none who is a member of the Akademie vir Wetenskap en Kuns. It will take time and conscious effort to establish our presence. The same is true of course for our universities. Again, with one exception, I know of no dean of a science faculty, vice-principal or principal who is a computer scientist. We consequently spend an enormous amount of time trying to explain the needs of computer science and its difficulties. I believe this symposium is a further step towards accreditation by our peers and superiors from the other sciences.

The total number of papers submitted to the Programme Committee for consideration was 34. Each paper was reviewed by three persons knowledgeable in the field it represents. Of those submitted, 23 were finally selected for inclusion in the symposium. As a result the overall quality of the papers is high and as a computer science community in Africa we can be justly proud of the final programme.

This is the fourth in the series of South African computer symposia. This year the symposium is sponsored by the Computer Society of South Africa (CSSA), the South African Institute for Computer Scientists and the local IFIP Committee. The executive director of the CSSA and his staff deserve warm thanks for handling the organisation as well as they have, while the Organising Committee provided Derrick and I with very valuable advice.

Finally I would like to express my sincere appreciation to the authors, to the members of the Programme Committee and particularly the reviewers. Without the kind cooperation of everyone, this symposium would not have taken place.

Pieter Kritzinger
SYMPOSIUM CHAIRMAN: PS Kritzinger, University of Cape Town.
SYMPOSIUM CO-CHAIRMAN: D Kourie, University of Pretoria.
MEMBERS OF THE PROGRAMME COMMITTEE

Judy Bishop, Witwatersrand University
Chris Bornman, UNISA
Hannes de Beer, Potchefstroom University
Gideon de Kock, Port Elizabeth University
Jaap Kies, Western Cape University
Derrick Kourie, Pretoria University
Pieter Kritzinger, Cape Town University
Tony Krzesinski, Stellenbosch University
Michael Laidlaw, Durban Westville University
Peter Lay, Cape Town University
Ken MacGregor, Cape Town University
Theo McDonald, Orange Free State University
Jan Oosthuizen, University of the North
Dennis Riordan, Rhodes University
Alan Sartori-Angus, Natal University
John Shochot, Witwatersrand University
Theuns Smith, Rand Afrikaans University
Trevor Turton, ISM (Pty) Ltd
Gerrit Wiechers, Infoplan.
LIST OF REVIEWERS

BERMAN Sonia
BISHOP Judy
BORNMAN Chris
CAREY Chris
CHERENACK Paul
DE BEER Hannes
DE VILLIERS Pieter
GORRINGE Pen
KIES Jaap
KOURIE Derrick
KRITZINGER Pieter
KRZESINSKI Tony
LAIDLAW Michael
LAY Peter
MacGREGOR Ken
MATTISON Keith
McDONALD Theo
RENNHACKKAMP Martin
RIORDAN Denis
SATORI-ANGUS Alan
SCHOCOT John
SMITH Theuns
TURTON Trevor
VAN DEN HEEVER Roelf
VAN ROOYEN Hester
VON SOLMS Basie
VOS Koos
# TABLE OF CONTENTS

**Keynote Address**

"An Extensible System and Programming Tool for Workstation Computers." ................. 1
Niklaus Wirth, ETH, Zurich

**Invited Lectures**

G Lasker, University of Windsor, Ontario.

D Teichrow, University of Michigan, U.S.A.

**Computer Languages I**

"SPS-Algol: Semantic Constructs for a Persistent Programming Language." ................. 13
S Berman, University of Cape Town.

"Petri Net Topologies for a Specification Language." .... 25
R Watson, University of the Witwatersrand.

"Towards a Programming Environment Standard in LISP." ... 45
R Mori, University of Cape Town

"ADA for Multiprocessors: Some Problems and Solutions." .. 63
J Bishop, University of the Witwatersrand.

**Computer Graphics**

"Polygon Shading on Vector Type Devices." ............... 75
C F Scheepers, CSIR.

"Hidden Surface Elimination in Raster Graphics Using Visigrams." ......................... 97
P Gorringe, CSIR.

**Database Systems I**

"On Syntax and Semantics Related to Incomplete Information Databases." .................. 109
M E Orlowska, UNISA.

"Modelling Distributed Database Concurrency Control Overheads." ......................... 131
M H Rennhackkamp, University of Stellenbosch.

**Operating Systems**

"The Development of a Fault Tolerant System for a Real-time Environment." .................. 149
M Morris, CSIR.

"A New General-purpose Operating System." .................. 161
B H Venter, CSIR.
Computer Languages II


"A Generalised Expression Structure." ...................... 189 W van Biljon, CSIR.

Computer Networks and Protocols I


Computer Networks and Protocols II


Artificial Intelligence

"A Data Structure for Exchanging Geographic Information." .......................................................... 267 A Cooper, CSIR.

"The Design and Use of a Prolog Trace Generator for CSP." ............................................................ 279 D G Kourie, University of Pretoria.

Database Systems II

"An Approach to Direct End-user Usage of Multiple Databases." ...................................................... 297 M J Phillips, CSIR.

"A Semantic Data Model Approach to Logical Data Independence." ..................................................... 329 S Berman, University of Cape Town.

Information Systems

"The ELSIM Language: an FSM-based Language for the ELSIM SEE." ..................................................... 343 L du Plessis and C Bornman, UNISA.

Computer Languages III

"Experience with a Pattern-matching Code Generator." ... 371
M A Mulders, D A Sewry and W R van Biljon, CSIR.

"Set-oriented Functional Style of Programming." ........ 385
C Mueller, University of the Witwatersrand.

Tutorial

The use of Modula-2 in Software Engineering." ............ 399
N Wirth, ETH, Zurich.
<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>07h30</td>
<td>Registration and Coffee.</td>
</tr>
<tr>
<td>08h45</td>
<td>Welcoming address, President of the South African Institute of Computer Scientists, Dr. G. Wiechers.</td>
</tr>
<tr>
<td>09h00</td>
<td>Invited Lecture, Professor D. Teichrow, University of Michigan.</td>
</tr>
<tr>
<td></td>
<td>Software Engineering, ... What Can We Expect in the Future.</td>
</tr>
<tr>
<td>10h00</td>
<td>COFFEE</td>
</tr>
<tr>
<td></td>
<td><strong>Computer Languages I.</strong> Chairman: G. Wiechers.</td>
</tr>
<tr>
<td>10h15</td>
<td>S. Berman, University of Cape Town.</td>
</tr>
<tr>
<td></td>
<td>SPS-Algo: Semantic Constructs for a Persistent Programming Language.</td>
</tr>
<tr>
<td>10h50</td>
<td>A. Watson, University of the Witwatersrand.</td>
</tr>
<tr>
<td></td>
<td>Petri Net Topologies for a Specification Language.</td>
</tr>
<tr>
<td>11h25</td>
<td>A. Mori, University of Cape Town.</td>
</tr>
<tr>
<td></td>
<td>Towards a Programming Environment Standard in USP.</td>
</tr>
<tr>
<td>11h50</td>
<td>J. Bishop, University of the Witwatersrand.</td>
</tr>
<tr>
<td></td>
<td>ADA for Multiprocessors: Some Problems and Solutions.</td>
</tr>
<tr>
<td>12h30</td>
<td>LUNCH</td>
</tr>
<tr>
<td>14h00</td>
<td><strong>Computer Graphics.</strong> Chairman: D. Kourie</td>
</tr>
<tr>
<td></td>
<td>C. F. Scheepers, CSIR.</td>
</tr>
<tr>
<td></td>
<td>Polygon Shading on Vector Type Devices.</td>
</tr>
<tr>
<td>14h35</td>
<td>P. Gorringe, CSIR.</td>
</tr>
<tr>
<td></td>
<td>Hidden Surface Elimination in Raster Graphics Using Visigrams.</td>
</tr>
<tr>
<td>15h15</td>
<td>COFFEE</td>
</tr>
<tr>
<td></td>
<td><strong>Database Systems I.</strong> Chairman: B. von Solms.</td>
</tr>
<tr>
<td>15h30</td>
<td>M.E. Orlowska, UNISA.</td>
</tr>
<tr>
<td></td>
<td>On Syntax and Semantics Related to Incomplete Information Databases.</td>
</tr>
<tr>
<td>16h05</td>
<td>M.H. Rennhadtkamp, Stellenbosch University.</td>
</tr>
<tr>
<td></td>
<td>Modelling Distributed Database Concurrency Control Overheads</td>
</tr>
<tr>
<td>18h00</td>
<td>Cocktail Party in Cullinan Room A.</td>
</tr>
</tbody>
</table>
## DAY 2

<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>08h30</td>
<td><strong>Keynote Address by Professor Niklaus Wirth, Swiss Federal Institute for Technology, Zurich.</strong></td>
</tr>
<tr>
<td></td>
<td><em>An Extensible System and a Programming Tool for Workstation Computers.</em></td>
</tr>
<tr>
<td></td>
<td><strong>Computer Networks and Protocols I.</strong> Chairman: P.S. Kritzinger.</td>
</tr>
<tr>
<td>09h30</td>
<td><strong>P.E. Krzesinski, University of Stellenbosch.</strong></td>
</tr>
<tr>
<td></td>
<td><em>An Approximate Solution Method for Multiclass Queueing Networks with State Dependent Routing and Window Flow Control.</em></td>
</tr>
<tr>
<td>10h05</td>
<td><strong>J. Punt, University of Cape Town.</strong></td>
</tr>
<tr>
<td></td>
<td><em>A Protocol Validation System.</em></td>
</tr>
<tr>
<td>10h30</td>
<td><strong>COFFEE</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Computer Networks and Protocols II.</strong> Chairman: A. van der Heever.</td>
</tr>
<tr>
<td>11h00</td>
<td><strong>P.S. Kritzinger, University of Cape Town.</strong></td>
</tr>
<tr>
<td></td>
<td><em>Protocol Performance using Image Protocols.</em></td>
</tr>
<tr>
<td>11h35</td>
<td><strong>Invited Lecture by Professor G. Lasker, University of Windsor, Ontario.</strong></td>
</tr>
<tr>
<td></td>
<td><em>The Relationship of Natural and Artificial Intelligence.</em></td>
</tr>
<tr>
<td>12h30</td>
<td><strong>LUNCH</strong></td>
</tr>
</tbody>
</table>

### Artificial Intelligence.
Chairman: G. Lasker.

<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>14h00</td>
<td><strong>A. Cooper, CSIR</strong></td>
</tr>
<tr>
<td></td>
<td><em>A Data Structure for Exchanging Geographic Information.</em></td>
</tr>
<tr>
<td>14h35</td>
<td>**A. I. Newcombe, University of Cape Town and R. Rado, National Library of Medicine, Maryland.</td>
</tr>
<tr>
<td></td>
<td><em>Strategies for Automatic Indexing and Thesaurus Building.</em></td>
</tr>
<tr>
<td>15h15</td>
<td><strong>COFFEE</strong></td>
</tr>
</tbody>
</table>

### Database Systems II.
Chairman: C. Bornman.

<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>15h30</td>
<td><strong>M.J. Philips, CSIR</strong></td>
</tr>
<tr>
<td></td>
<td><em>An Approach to Direct End-user Usage of Multiple Databases.</em></td>
</tr>
<tr>
<td>16h05</td>
<td><strong>S. Berman, University of Cape Town.</strong></td>
</tr>
<tr>
<td></td>
<td><em>A Semantic Data Model Approach to Logical Data Independence.</em></td>
</tr>
<tr>
<td>16h45</td>
<td><strong>Open Forum with professors G. Lasker, D. Teichrow and N. Wirth.</strong></td>
</tr>
<tr>
<td></td>
<td>Moderator: Dr. D. Jacobson.</td>
</tr>
<tr>
<td>19h30</td>
<td><strong>Symposium Banquet in Cullinan Room.</strong></td>
</tr>
<tr>
<td></td>
<td>Guest speaker, Dr. D. Jacobson, Group Executive: Technology, Allied Technologies Limited.</td>
</tr>
</tbody>
</table>

### Information Systems.
Chairman: D. Teichrow.

<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>L. du Plessis and C. Bornman, UNISA.</strong></td>
</tr>
<tr>
<td></td>
<td><em>The ELSIM Language: an FSM-based Language for the ELSIM SEE.</em></td>
</tr>
<tr>
<td>14h00</td>
<td><strong>J. Mende, University of the Witwatersrand.</strong></td>
</tr>
<tr>
<td></td>
<td><em>Three Packaging Rules for Information System Design.</em></td>
</tr>
<tr>
<td>15h15</td>
<td><strong>COFFEE</strong></td>
</tr>
</tbody>
</table>

### Computer Languages III.
Chairman: N. Wirth.

<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>W. van Biljon, CSIR.</strong></td>
</tr>
<tr>
<td></td>
<td><em>Experience with a Pattern-matching Code Generator.</em></td>
</tr>
<tr>
<td></td>
<td><strong>C. Mueller, University of the Witwatersrand.</strong></td>
</tr>
<tr>
<td></td>
<td><em>Set-oriented Functional Style of Programming.</em></td>
</tr>
</tbody>
</table>

### Notes:
- **Computer Networks and Protocols I.**
- **Computer Networks and Protocols II.**
- **Database Systems II.**
- **Computer Languages III.**
<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>08h00</td>
<td>Registration (Tutorial only).</td>
</tr>
<tr>
<td>08h30</td>
<td>Tutorial.</td>
</tr>
<tr>
<td></td>
<td>The Tutorial will be given</td>
</tr>
<tr>
<td></td>
<td>by professor Niklaus Wirth,</td>
</tr>
<tr>
<td></td>
<td>Division of Computer Science,</td>
</tr>
<tr>
<td></td>
<td>Swiss Federal Institute of</td>
</tr>
<tr>
<td></td>
<td>Technology, Zurich.</td>
</tr>
<tr>
<td></td>
<td>The use of Modula-2 in</td>
</tr>
<tr>
<td></td>
<td>Software Engineering.</td>
</tr>
<tr>
<td></td>
<td>Topics to be covered include:</td>
</tr>
<tr>
<td></td>
<td>What is Software Engineering?</td>
</tr>
<tr>
<td></td>
<td>Data types and structures.</td>
</tr>
<tr>
<td></td>
<td>Modularization and</td>
</tr>
<tr>
<td></td>
<td>information hiding.</td>
</tr>
<tr>
<td></td>
<td>Definition and implementation</td>
</tr>
<tr>
<td></td>
<td>parts.</td>
</tr>
<tr>
<td></td>
<td>Separate compilation with</td>
</tr>
<tr>
<td></td>
<td>type checking.</td>
</tr>
<tr>
<td></td>
<td>Facilities to express</td>
</tr>
<tr>
<td></td>
<td>concurrency.</td>
</tr>
<tr>
<td></td>
<td>Pompous programming style.</td>
</tr>
<tr>
<td></td>
<td>What could be excluded?</td>
</tr>
<tr>
<td>12h15</td>
<td>Close of Symposium.</td>
</tr>
<tr>
<td>12h30</td>
<td>LUNCH</td>
</tr>
</tbody>
</table>
A SEMANTIC DATA MODEL APPROACH TO LOGICAL DATA INDEPENDENCE

Sonia Berman

University of Cape Town

ABSTRACT

Semantic data models were originally introduced as a tool for data description and design. SDMI (Semantic Data Model Interface) investigates their use in another role — as the basis for a data manipulation language — and shows how this can provide a means of achieving complete logical data independence. With SDMI users view a database simply in terms of objects and their attributes, and can access any combination of related information just by stating the relationship(s) of interest. This paper outlines the problem of logical data dependence that is inherent in all existing database systems and shows how the universal relation approach fails to provide a complete solution. SDMI is described and shown to be a more promising alternative.
INTRODUCTION

Relational databases were introduced to simplify database programming by making the programmer immune to the data structures chosen for storing the data [6]. However they fail to achieve this completely, as they provide only physical data independence. The user still needs to know the detailed logical organisation of the database: what relations exist, the attributes they comprise, and what attributes can be matched in a join operation. This is not adequate, especially if a database is large or subject to restructuring. Providing views does not alleviate the problem - view names and attributes must still be known, and there are often even more views than base relations. Moreover since the logical structure is not transparent to programmers, databases cannot evolve freely without affecting existing applications.

The universal relation is an attempt at rectifying this situation[8]. However it presents several problems itself, with the result that Kent summarised its contribution as follows: "the universal relation assumption undoubtedly does solve some problems in relational database theory. Those problems ought to be identified and reexamined, and solved in a better way" [6]. This paper describes such a re-examination and an alternative solution to providing true data independence.

SDMI is an interface to a relational database which presents the user with a simple, natural view of data as a collection of objects described by attributes. Any combination of attributes can be accessed together, without the need for a user to "navigate" through the database. Instead the simple concept of a mapping is employed: a precise, unambiguous and natural way to specify data relationships. As an example "Project.Leader.Name" indicates the Name of the Leader of the Project. The inter-relation joins required to follow this sequence of associations is automatically performed by the SDMI system and is completely transparent to the user.

The relational model has also been criticised [5] for not directly supporting the notion of entity (object), generalisation/specialisation [10] or grouping [4]. Entities in the real world do not map one-to-one onto database constructs, but onto several different relations or subrelations. With generalisation the common properties of similar objects are extracted to define a more general object type such that each of the original "is-a" (specialised) instance of this. At the level of the generic object, differences are suppressed and shared properties emphasized. The specialisations of a generic object inherit its attributes in addition to their own. Grouping allows a set of objects to be viewed as a higher-level object. Codd attempted to
incorporate entities, generalisation and groupings in the RM/T extension to the relational model [2], but the result is complex and awkward to use. Since SDMI was developed from the SDM model [4] which is based on exactly these three concepts, it encompasses these notions in its interface in a much simpler and cleaner fashion.

This paper describes the SDMI data manipulation language and explains how it is translated into equivalent relational operations. The incorporation of semantic abstractions - generalisation/specialisation and grouping - and the facilities required to support these constructs are outlined. The universal relation approach to logical data independence is then described and SDMI is shown to be free of the problems inherent in this technique. Finally, the advantages of SDMI as a database language are discussed and some suggestions for future work made.

THE SDMI LANGUAGE

The SDMI data model is based on Hammer and McLeod's SDM[4], extended to include functional dependencies (FDs). A schema comprises a list of classes (object types), each described by several attributes (properties and relationships of the object). A class can be either base, or a grouping (set) or "subclass" (specialisation) of another class. In figure 1 Workerscomm is a grouping, Newemp and Senioremp are subclasses; the remaining declarations are base classes.

The SDMI language is non-procedural, relationally complete [3] and provides associative retrieval on any attribute. The commands are GET, CREATE, DELETE, UPDATE, INSERT, REMOVE and NULLIFY (see figure 2). The latter is a special form of UPDATE which allows individual values of a multivalued attribute to be deleted, leaving other values of that attribute intact. Unlike CREATE and DELETE which add and destroy data, INSERT and REMOVE affect only generalisations and groupings. For example, having CREATED a EMPLOYEE occurrence one could subsequently issue two INSERT commands to include it in the subclass SENIOREMP and a WORKERSCOMM grouping. If it were later REMOVED from the workerscommittee, it would remain on the database until DELETED. GET can be used to retrieve metadata from the database, and the option of file input/output is provided for CREATE, GET and UPDATE.
DATABASE factory
CLASS employee
ATTRIBUTES
  Name : Char[30];
  Wage : Int;
  Dept : Department;
  Position : Char[8];
  Status : Char[8];
  Years-Served : Int;
  Skills : Char[15] multivalued;
  Project : Job;
KEYS Name;
FDS Wage -> Position;
CLASS department
ATTRIBUTES
  Name : Char[20];
  Mgr : Employee;
  Projects : Job multivalued;
  Size : Int;
  Location : City;
KEYS Name, Mgr;
CLASS workerscomm GROUP OF employee
ATTRIBUTES
  Chairman : Employee;
  Next-Meeting : Int;
CLASS newemp ISA employee WHERE Years-Served < 2
ATTRIBUTES
  Comments : Char[24] multivalued;
  Grading : Int;
  Increase : Int;
  FDS Grading -> Increase
CLASS seniorememp ISA employee
ATTRIBUTES
  Lastincrease : Int;
CLASS City
ATTRIBUTES
  Name : Char[15];
  Distance : Int;
CLASS Job
ATTRIBUTES
  Leader : Employee;
...
...

Figure 1. Part of an SDMI schema
CREATE employee(Name = "Doe", Dept.Name = "Toy", Wage = 1250)

UPDATE employee(Wage = Wage * 1.15)
  WHERE Dept.Name = "Toy"

DELETE employee WHERE Status = 'Poor' AND Dept.Size > 20

GET employee( cname = Name, cwage = Wage, cmgr = Dept.Mgr.Name)
  WHERE Wage > (0.8 * Dept.Mgr.Wage)

NULLIFY department( Projects )
  WHERE Name = "Toy" AND Projects.Type = "Robots"

INSERT employee INTO senioremp WHERE Name = "Fig"

REMOVE employee FROM workerscomm WHERE Name = "Fig"

Figure 2. Some example SDMI statements.

The attribute is the unit of access, so the ordering of attributes within a class is not significant. Commands operate on sets of objects identified by a selection condition expressed in terms of any attributes or relationships of that class. Arithmetic operations and aggregate functions (average, minimum, maximum, sum and count) can be applied to database values, and any desired sorting of results can be specified.

The SDMI notation enables multivalued attributes, and attributes of attributes, to be referenced easily and concisely. Consider the following statement for the schema in figure 1:

CREATE department(Name = "Videos"; Projects.Leader.Name = "Doe", "Li", "Fig"; Size = 3)

Projects is a multivalued attribute of department which is here being given three values. "Projects.Leader.Name" is a "mapping" [4], which permits attributes of attributes to be designated simply. It enables the relationship between the department and the strings "Doe", "Li", "Fig" to be precisely defined in a natural and succinct manner. If two or more attributes are needed to uniquely identify an object such as Leader they are given jointly, as in:

CREATE department(Name = "Videos"; Projects.Leader.Name, Projects.Leader.Position = "Doe", "Salesman", "Li", "Salesman", "Fig", "Head"; Size = 3)

SDMI commands are designed to be embedded within a C [7] host language program. As in most database systems, a preprocessor replaces SDMI statements within a C program by
appropriate commands manipulating the underlying database. Each SDMI statement is generally translated into several such commands because the attributes can be scattered over several relations, and subclass and grouping membership may require adjustment to maintain consistency.

IMPLEMENTATION

In what follows, attributes of relations will be referred to as "columns", to distinguish them from the attributes in the SDMI schema.

A schema processor compiles every schema, designs its underlying relations and stores the metadata in a new database. Relation schemas are designed in the following way[1]. In general one relation is created for each class, comprising all the single-valued attributes of that class. A separate relation is created for each multivalued (repeating) attribute, and additional relations are generated for grouping classes. As an example relation (WORKERSCOMM, CONTENTS) enables several employees to be stored with each workerscomm group. Functional dependencies are examined and the relations adjusted if necessary to ensure the resulting scheme is in 3rd Normal Form.

Relation ISA (SUPERTYPE, SUBTYPE), with its attributes drawn from the domain of relation names, has a tuple for each superclass-subclass relationship. All generalisations of a subclass are stored here so that all its superclasses can be determined in one simple operation, which is essential for processing inherited attributes (see later). For every subclass X defined on a class Y, an additional (boolean) column X is created in the relation for Y, indicating subclass participation of Y occurrences in X (see Newemp, Senioremp in fig. 3).
Surrogates are used to uniquely identify objects in the database, as in Codd's RM/T model [2]; the domain of any non-printable attribute is therefore a surrogate domain. These surrogates are transparent to the user, who references a specific class by supplying a value for any of its identifiers. This has the advantage that access is not restricted to one "primary" key; it also facilitates joins and saves space in the database. There are generally several relations that together form the representation of a class; each such relation includes one "class$" (surrogate) column.

The major problem in providing high-level access to a relational database lies in the ability to correctly relate columns from different relations. Automatic navigation between relations in this way is made possible in SDMI through the use of surrogates, because each relation includes a column of surrogates of the one (and only one) class that it describes. As that surrogate column appears in all relations that reference this class, it is possible to join any two such relations automatically, as explained below.
If an attribute is not in the same relation as the other attributes of its class the attribute is multivalued, or it is inherited from a superclass, or it was removed because of a violation of 3rd Normal Form. Any relation created for a multivalued attribute has two columns, one representing the attribute, and the other being the surrogate of the class it characterises (eg. RSKILLS in fig. 3 can be joined to EMPLOYEE on their common Employee$ column). Surrogates are only defined for base classes and grouping classes; subclasses are represented by the surrogate of the base class from which they are derived. In this way subclass tuples will always include the same surrogates as the tuples containing their inherited attributes. Hence any category of employee can always be cross-referenced to any other category of employee directly, since they all use Employee$ surrogates. If say the Name and Grading of a Newemp is requested (Newemp is-a Employee), the Employee$ surrogates can be used to join the Newemp relation containing Grading to the Employee relation containing Name. (See fig. 3) An attribute placed in a separate relation because of an FD is always accessible via the left hand side attribute(s) which appear in both relations (eg. Wage can be used to join RPOSITION and EMPLOYEE).

The above discussion showed how any combination of attributes of the same class can be correctly retrieved. When a statement involves attributes of different classes, the objects must be related by mappings. The greatest use of surrogates to join relations occurs when processing mappings. In general, a mapping of the form A.B implies that the domain of attribute A is some class C1 (rather than a printable value) and that B is an attribute of C1. Suppose that A is an attribute of class C2. The linking of C2 entities with the appropriate C1 objects is achieved via attribute A which will be implemented as a column of C1 surrogates. No matter what relation B values are stored in, it will always have a C1$ column to join on, because it is a relation describing C1 objects. In this way any valid mapping can be supported in SDMI, making mappings a simple and powerful alternative to database navigation. (Consider eg. Dept.Name and Dept.Mgr.Name of say "Doe" in fig. 3)

SDMI metadata includes the name of the column and relation representing each attribute in the schema. Thus a request for data, which is given in terms of the user's high-level (semantic) view of the database, is easily translated into a list of target columns. Selecting the correct target column is thus straightforward, except in the case of a query such as that in figure 4. The set of classes having attribute Name does not include Newemp. This indicates that Name is an inherited attribute of Newemp, and one (and only one) of these classes is the superclass from which it inherits Name. To establish which, it is necessary to know all ancestors of
Newemp in the generalisation hierarchy, as Name could potentially be associated with any of these. By referencing the ISA relation, all of these can be accessed directly, and the target relation determined accordingly.

GET newemp(cname = name) WHERE Grading == "B"

<table>
<thead>
<tr>
<th>ISA</th>
<th>Supertype</th>
<th>Subtype</th>
</tr>
</thead>
<tbody>
<tr>
<td>employee</td>
<td>newemp</td>
<td></td>
</tr>
<tr>
<td>employee</td>
<td>senioremp</td>
<td></td>
</tr>
<tr>
<td>department</td>
<td>top-dept</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>META</th>
<th>Attribute</th>
<th>Class</th>
<th>Relation</th>
<th>Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>employee</td>
<td>employee</td>
<td>name</td>
<td>-</td>
</tr>
<tr>
<td>name</td>
<td>department</td>
<td>department</td>
<td>name</td>
<td>-</td>
</tr>
<tr>
<td>name</td>
<td>city</td>
<td>city</td>
<td>name</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 4. Support for attribute inheritance.

GENERALISATION AND GROUPING

In addition to "user-controllable" groupings, where insertion is explicitly performed by the user, SDMI supports two types of grouping class [4] for which membership is automatically maintained by the system.

(i) To find the constituents of a grouping defined "on common value of" some attribute, eg PROJGROUPS comprising employees with common Project values, appropriate tuples of the EMPLOYEE relation are selected according to the Project of interest.

(ii) The instances of an enumerated grouping class comprise the subclass sets listed, eg the enumeration "NEWEMPS, MANAGERS and SENIOREMPS" will involve exactly three EMPLOYEE groupings viz. the set of NEWEMPS, MANAGERS and SENIOREMPS respectively. These groupings are derived in a similar fashion to those defined "on common value" of an attribute, by testing subclass participation flags.

Inclusion of objects in subclasses is also performed automatically according to the subclass definition[4], if not user-controllable. With a subclass defined as the intersection, complement or union of other subclasses this is achieved by ANDing, negating or ORing the relevant subclass indicators. A subclass defined in terms of relationship participation, such as Manager "where is a value of Mgr of DEPARTMENT" has its membership flag set by examining the Mgr column. Whenever any Mgr value is altered, the system adjusts the subclass membership of the employees involved accordingly. Such consistency maintenance is outlined in the
Predicates that define subclasses, such as NEWEMP is-a EMPLOYEE "WHERE Years-Served < 2", can be of arbitrary complexity and can potentially be expressed in terms of several attributes. Hence if participation in such a subclass is to be maintained by the system, there is the overhead of recomputing the predicate value to check subclass membership whenever any of these attributes is updated. For this reason the user is informed that such a subclass should be defined as user-controllable instead, when this overhead is not justified.

**SUPPORTING THE SEMANTICS**

Dependency information is kept to enable participation in subclasses to be maintained consistently. Metadata relation AFFECTS has columns Agent and Dependant, comprising the attribute names and subclass names appearing in a subclass definition, respectively. Several tuples may be required for a subclass definition, because of mappings. Consider a simple example: TOP-DEPT is-a DEPARTMENT "WHERE Mgr.Status = 'Good'". If a department's Mgr is altered, that department's subclass membership must be re-examined. If any manager's Status is changed, all departments with the corresponding Mgr must have their membership adjusted. Thus tuples are entered in AFFECTS for each component of a mapping. If a subclass A is defined in terms of B.C, two tuples with (Dependant A and) Agent B and C respectively are used.

Whenever the value of any attribute B is altered by the user, all subclasses dependant on B are determined from AFFECTS, and their membership recomputed. This process uses the stored metadata which includes an encoded version of subclass definitions.

Predicates defining subclasses can include mappings of arbitrary length. A separate relation Mapping, with Mapping surrogates to distinguish its tuples, is used to store these. (A similar relation EXPR stores expressions that occur in subclass predicates.) Each mapping tuple comprises: mapping surrogate, position in mapping, relation and column representing that component, and surrogate of the class described by that relation. A mapping such as Dept.Location.Name would be represented as below:

(123, 1, EMPLOYEE, Dept, Employee$)
(123, 2, DEPARTMENT, Location, Department$)
(123, 3, CITY, Name, City$).

For each tuple in turn the system generates "==<relation>..<surrogate> and <relation>..<column> ". Thus the above would become

<x> == EMPLOYEE.Employee$ and EMPLOYEE.Dept == DEPARTMENT.Department$ and DEPARTMENT.Location == CITY.City$ and CITY.Name <op> <z>
(Note that x, op and z come from the context in which the mapping is used. For example, in "Delete Newemp where Dept.Location.Name == 'PE'", the initial "NEWEMP.Employee" and the final "== 'PE'" would complete the expression.)

THE UNIVERSAL RELATION APPROACH

A universal relation scheme (u.r.s.) also permits users to access a database without being aware of its structure, although it does not incorporate generalisation or grouping. It is based on the assumption that the same column name always plays the same role wherever it appears in a database. Information in separate relations can only be linked through identical column names. The u.r.s. allows any combination of attributes in a query, without any need to specify access paths eg.

GET employee, project WHERE department == "Toy".
The query is interpreted by taking the union of all natural joins of relations containing employee, project or department, joining on columns with the same name.

Unfortunately this only permits navigation along certain paths in the schema; something other than the simplest, direct relationship between two attributes cannot be determined. Thus any query involving Employee and Manager will retrieve the manager of an employee, never the manager of the manager of an employee, nor any other association between them. In SDMI any relationship such as Mgr.Mgr.Name can always be specified in a mapping.

A u.r.s. can cause problems when different names are used to reference the same entity, eg. Employee and Manager [9]; if Manager appears in relation DEPT the attributes of managers such as their wage cannot be determined from the EMPLOYEE relation because a Manager column cannot be matched with an Employee column in a join. The u.r.s. has no way of "knowing" that managers and employees represent the same entities. This problem does not arise in SDMI, as the name is accessible through the mapping Mgr.Name.

The u.r.s. further assumes that for every set of attributes there is one unique association that the user has in mind. Two relationships such as ASSIGNED(Employee,Project) and LOANED(Employee,Project) cannot both be supported because of this "unique role assumption" [9]. Their union will always appear in any query involving employees and projects; it is impossible to extract either relationship on its own. A user can easily misinterpret the result of a query because of insufficient knowledge, eg. if he is unaware of loans when accessing Employee and Project he will think they are all assignments. This occurs because the meaning of a query is inferred by the system; the user cannot indicate what is intended. To avoid these problems the database designer must use distinct names such as Assigned-Proj and Loaned-
Proj, but then these will never be equated with a "Project" column in a join. In SDMI an object can have any number of attributes of the same type, each playing a different role to identify different associations, distinguished by their names. Unlike the u.r.s. the names cause no problems with accessing further information because of mappings eg. Assigned-Proj.Leader, Loaned-Proj.Duedate. There is no possibility of queries being misinterpreted, because the user specifies exactly what relationships are of interest in his mapping.

Choosing column names for a u.r.s. is a nontrivial problem [6]. Consider a database with a relation R(Emp, Assigned-Proj, Loaned-Proj) requiring a new relation to store project managers. It is generally difficult to decide whether this should be R2(Assigned-Proj,Mgr) or R2(Loaned-Proj,Mgr), and in either case the first attribute has a strange name. A separate "domain" column Project in addition to the two in R may also be necessary, to cater for projects with no employees but with other data of interest eg Fees. There is no corresponding problem in SDMI where Assigned-Proj, Loaned-Proj and Project can be equated because they are of the same type, irrespective of their names.

The u.r.s. involves extensive use of null values. This arises from the fact that columns actually represent domains and hence any two columns of the same name must contain identical sets of values at all times [6]. Thus in a relation for employees and books on loan, the Employee column must include all employees and the Book column every book (the vast majority of tuples having a null Employee or Book value)! The use of nulls means that dependencies must be restricted tononnull left hand side attributes and the notion of a key must be relaxed to allow nulls, otherwise the u.r. could have no key[6]. There is no such need for nulls in SDMI because attributes can map onto a subset of their domain.

EVALUATION

The major advantages of SDMI over existing relational languages are that several underlying relations can be referenced in one statement as if the information were all stored together, and that objects can be uniquely distinguished via any candidate key (see fig. 5). Total data independence is achieved in SDMI, an improvement on the relational system where users need to know how items are grouped to form relations, to know when a join is required and which columns to equate in the join. Unlike the u.r.s. approach, problems with multiple roles, column names, nulls and indirect associations are avoided. In addition, SDMI supports grouping, subclasses and inherited attributes.

SDMI schemas are easy to develop, unlike those of conven-
tional databases where a semantic model is only an inter-
mediate step in the design process. In SDMI there is no 
need for the additional step mapping this onto a data 
structure. The metadata relations created by the system are 
accessible to programmers, enabling them to query the data 
definition in the same way as the data content.

SDMI has shown that a database interface based on a semantic 
data model can provide complete data independence, enhancing 
ease of use and making programs immune to changes in the 
database structure. It is intended that the system be 
extended to include security constraints, database transac-
tions and data derivations.

(a) SELECT first.name, first.wage, 
FROM employee first, employee second, department 
WHERE first.wage > (0.8 * second.wage) 
AND first.dept = department.name 
AND department.mgr = second.name

(b) GET (cname = employee.name, cwage = employee.wage, cmgr 
= employee2.name) 
WHERE employee.wage > (0.8 * employee2.wage) 
AND employee.deptno = department.deptno 
AND department.mgr = employee2.empno

(c) GET employee (cname = Name, cwage = Wage, cmgr = 
Dept.Mgr.Name) 
WHERE Wage > (0.8 * Dept.Mgr.Wage)

Figure 5. Comparison with popular relational languages. 
(a) The SQL and (b) the EQUEL (Ingres) statements 
equivalent to (c) an SDMI GET command.

REFERENCES

Automated Database Design Tool". Inform. Syst. 11(2), 149-
165 (1986).

Capture More Meaning". ACM Trans. on Database Syst. 4, 397-
434 (1979).

Addison-Wesley (1977).

SDM: A Semantic Database Model". ACM Trans. on Database 
Syst. 6(3), 351-386 (1981).

[5] W. Kent. "Limitations of Record Based Information


