WHY ROSE IS FAST: FIVE OPTIMIZATIONS IN THE DESIGN OF AN EXPERIMENTAL DATABASE SYSTEM FOR CAD/CAM APPLICATIONS

Martin Hardwick
Department of Computer Science
and Center for Interactive Computer Graphics
Rensselaer Polytechnic Institute

hardwick@csv rpi edu
518-271-2712

Abstract

ROSE is an experimental database system for CAD/CAM applications that organizes a database into entities and relationships The data model of ROSE is an extension of the relational model and the data manipulation language is an extension of the relational algebra Internally, ROSE is organized so that it can use operating system services to implement database system services In this paper we describe five optimizations that have helped to make ROSE a fast database system for CAD/CAM

1 Introduction

On one level the answer to the question posed by the title of this paper is simple ROSE is fast because CAD/CAM applications need to be fast [Sid80] On another level the answer is more complicated because fast general purpose CAD/CAM database systems are difficult to implement [Kat85]

ROSE is an acronym for the Relational Object System for Engineering The word "Relational" is important here because ROSE has a data model that is a superset of the relational model and a data manipulation language that is a superset of the relational algebra This feature of ROSE may be surprising because engineering "folklore" states that relational databases are too slow to be used in CAD/CAM

In the acronym the word object is used in the sense of Complex Object database systems [Lor83], not in the sense of object oriented programming languages [Dit86] For this reason in this paper we will use the term entity to describe units of data that might elsewhere be called objects [Spo86] ROSE models a database as sets of entities where each entity can be a complex data aggregate The important features of ROSE are summarized below

1 It models databases using the Entity-Relationship model [Che76]

2 It describes entity data structures using AND/OR trees [Mcl83]

3 It stores entities as files in operating system directories

4 It manipulates entities using an entity algebra that is an upward compatible extension of the relational algebra [Har87]

5 It caches entities in a main memory workspace while they are being edited

In each of the next five sections we give one reason why ROSE is fast In the sixth section we outline a typical ROSE application and in the last section we give some performance statistics that demonstrate that ROSE is fast in an "artificial" application

2. Entity Data Structures

The first reason for the speed of ROSE is stated below

> 1 The data in an entity can be found in a single search

In ordinary relational databases the data in a CAD/CAM entity has to be distributed between many tuples Therefore, this data must be retrieved in several searches (using joins) The time to perform these searches may not be significant if all of the tuples are clustered into the same disk block [Lor83, Wil85], but this is difficult to achieve when different applications want to cluster data in different ways In Section 4 we discuss how ROSE manages multiple applications that share the same data In this section and the next section we show how ROSE manages the data of a single application

ROSE defines the data structure of an entity using an AND/OR tree [Mcl83] Figure 1 shows how two AND/OR trees can be used to define the data structures of two simple entities The first tree in the figure defines a domain for *point* entities using an AND node The second tree defines a domain for *number* entities using an OR node The distinction between the trees is shown by an arc joining the downward edges of the AND node Upper case names

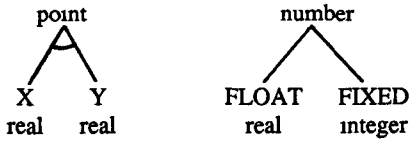

Figure 1 AND/OR Trees for Points and Numbers

represent attributes and lower case names represent domains

An AND node defines a domain so that an entity in that domain must contain a sub-entity in *every one* of its attributes An OR node defines a domain so that an entity in that domain must contain a sub-entity in *exactly one* of its attributes [Smi77] Hence

i A *point* entity must be created by abstracting a *real* entity with the attribute X and a *real* entity with the attribute Y

ii A *number* entity must be created by abstracting either a *real* entity with the attribute FLOAT or an *integer* entity with the attribute FIXED

We will use an AND/OR tree to describe a data structure for circuit logic entities A circuit logic entity describes a circuit as a list of sub-circuits connected by wires In our database, the description of a circuit will be broken into two parts The *interface* part will describe the external pins of a circuit and the *version* part will describe an implementation of the circuit. Each *interface* may be associated with multiple *versions* because a circuit can be implemented in many different ways [Bat85] We will describe a data structure for *version* parts and assume that the reader can invent a suitable structure for *interface* parts

Figure 2 contains an AND/OR tree for *version* entities The tree makes a *version* entity contain an INAME entity and a CONTENTS entity The INAME entity is used to contain the key of an *interface* entity (The meaning of this key will be described in the next section) The CONTENTS entity contains a list of entities, where each entity describes a SOCKET, a GATE or a WIRE in the circuit The list property is shown by a star (*) below the origin of the *contents* node This property allows the entities in a domain to contain any number of OR abstractions if the node is an OR node, and any number of AND abstractions if the node is an AND node

A SOCKET entity describes a sub-circuit with a known interface (specification) but an unknown version (implementation) [Bat85] A GATE entity defines a sub-circuit with a known interface and version Both types of entities have a data structure defined by the *cell* domain This structure contains "slots" for the key of an interface or version, the coordinates (LL and UR) of a bounding box surrounding the entity, and an identifier that distinguishes this entity from all of the other sub-circuits in a circuit

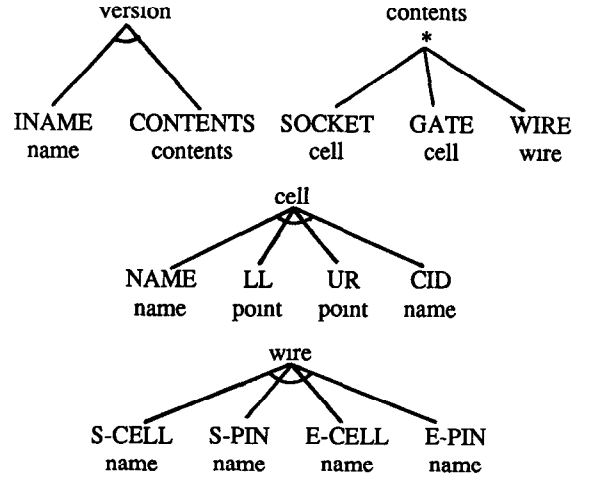

Figure 2 Structure Model for Circuit Logic Entities

A WIRE entity connects two pins in a circuit These pins are described by the interface of a GATE or SOCKET, or by the interface of this circuit If we call a GATE or SOCKET a cell, then a wire entity contains "slots" for the ID's of two cells and two pins The pin ID's are represented by pin NUMBERs that must be unique within a particular cell The cell ID's correspond to the CID attribute of a cell entity By convention, no cell is given a CID value of zero because this value is used to show that a wire is linked to an external pin of the circuit

Figure 3 describes an entity belonging to the *version* domain This entity is a four input AND gate that has been implemented by instantiating and connecting three, two input AND gates The circuit does not contain any SOCKET entities because the designer has been able to select *versions* for each of these gates The entity is described using a modified LISP notation The notation gives an entity or nested sub-entity a left parenthesis, an attribute, a data value and a right parenthesis If the data value is not primitive, then it will be made up of nested sub-entities described using the same notation

Although the notation shown in Figure 3 is easy to use, it has the unfortunate property of hiding the relationship between ROSE and relational databases Nevertheless, we hope the reader can see that a ROSE database containing one level AND nodes only is an unusual type of relational database, and a ROSE database containing AND nodes nested to any depth is an unusual type of Non First Normal Form (NFNF) relational database [Dad86]

3 Entity Keys

In ROSE the name of an entity is a key that can be used to find that entity [Sto83] Our second optimization makes the search for this entity more efficient

2 After the first search, the physical address of an entity is stored as a hidden field within the key used to select that entity

This optimization requires the address of an entity to be constant for the duration of a design session It does not work until after the first search because not every *name* value stored in a database will be used to find entities Therefore, ROSE does not convert the logical address given by a name into a physical address until a request is made by the user Furthermore, in ROSE entities in different entity sets can have the same name so the system is careful to make sure that an optimized search refers to an entity in the same entity set

Figure 4 contains a modified Entity-Relationship (E-R) diagram for a database that makes significant use of this optimization This diagram divides a logic circuit into CELLs, WIREs and PINs so that these entities can be edited and manipulated, individually As in a normal E-R diagram, Figure 4 represents entity sets by square boxes and relationship sets by diagonal boxes However, unlike a normal E-R diagram the rounded boxes describe the key of an entity, they do not describe the attributes of an entity because these attributes are described in a separate AND/OR tree

```
(VERSION (INAME 4and)
   (CONTENTS (GATE (NAME and)
                   (LL (X 1 0) (Y 1 0))
                   (UR (X 3 0) (Y 3 0))
                   (CID 1))
             (GATE (NAME and)
                   (LL (X 1 0) (Y 5 0))
                   (UR (X 3 0) (Y 7 0))
                   (CID 2))
             (GATE (NAME and)
                   (LL (X 5 0) (Y 3 0))
                   (UR (X 7 0) (Y 5 0))
                   (NAME and)
                   (CID 3))
             (WIRE (S-CELL 1) (S-PIN 3)
                   (E-CELL 3) (E-PIN 1))
             (WIRE (S-CELL 2) (S-PIN 3)
                   (E-CELL 3) (E-PIN 2))
             (WIRE (S-CELL 0) (S-PIN 1)
                   (E-CELL 1) (E-PIN 1))
             (WIRE (S-CELL 0) (S-PIN 2)
                   (E-CELL 1) (E-PIN 2))
             (WIRE (S-CELL 0) (S-PIN 3)
                   (E-CELL 2) (E-PIN 1))
             (WIRE (S-CELL 0) (S-PIN 4)
                   (E-CELL 2) (E-PIN 2))
             (WIRE (S-CELL 3) (S-PIN 3)
                   (E-CELL 0) (E-PIN 5))))
```

Figure 3 A Version Entity

The key of an entity is treated specially because in CAD an entity's keys is usually not an intrinsic part of its data value Instead, it has a role with respect to the entity similar to the role of a file name to a file in a text processing system That is, it is an identifier invented by the user or the operating system to distinguish this entity from the other entities in an entity set In ROSE the key of an entity is used to implement relationships because if there is a relationship from entity set A to entity set B, then key values from B will be stored in A The degree of a relationship (one to one, one to many, etc) can be controlled by making room for the appropriate number of "slots" in an entity's data structure

The attributes of the VERSION, GATE, SOCKET and WIRE entity sets in Figure 4 were described in Figure 2 The attributes of PIN and INTERFACE entities will not be described here Note that every entity in the modified E-R diagram must have a data structure defined by an AND/OR tree, but the opposite is not true because a node in an AND/OR tree need not correspond to an entity set in the E-R diagram If this is the case, then entities in this domain cannot be stored in a database unless they are first integrated into a higher level entity For example, a POINT entity must be integrated into a SOCKET or GATE entity before it can stored in the database of Figure 4

4 The Entity Algebra

When a circuit is being edited it is divided into WIREs, CELLs and PINs so that each of these features can be edited and manipulated, individually When a circuit is used as a sub-circuit it is stored as a single indivisible unit of data (in the VERSION entity set) so that its "design space" will not conflict with the "design space" of the circuit that it being edited Hence, the third reason for the speed of ROSE is as follows

3 Entities can be translated between data structures in an application independent way

This advantage supplements the first because, in principle, it means that every application can have an organization that allows the data in an entity to be found in a single search We qualify this statement with the phrase "in principle" because some CAD applications need extremely complex databases and the best way of organizing these databases is not yet clear [Afs85] For example, consider a circuit design application where a circuit can have behavior, layout and logic descriptions

There are two reasons why ROSE is able to tailor the data structure of an entity to its current application The first is the nature of CAD applications [Kat85] The typical CAD transaction corresponds to an engineering design session Therefore, it takes place over an extended period of time and the overhead of translating data structures at the start and end of a transaction is not excessive The second, more intrinsic, reason is ROSE's entity algebra

The entity algebra is an extension of the relational algebra It contains functions that can be used to create

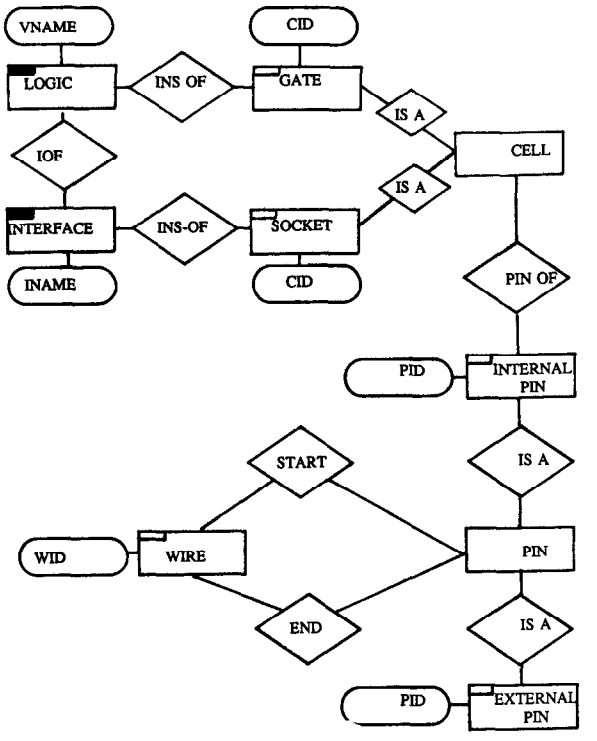

Figure 4 Storage Model for Logic Circuits

entities out of tuples and vice versa For example, a *version* entity can be created out of a tuple with INAME and CONTENTS attributes In ROSE a new type of relation has been invented so that it can be used to create entities with a data structure defined by an OR node A relation of this type is defined so that its tuples must contain a value in *exactly one* of their attributes The other attributes must all be empty (contain null values)

As an illustration, the first expression below creates a point entity out of a tuple with X and Y attributes, and the second expression creates two number entities out of two tuples with FIXED and FLOAT attributes In the first expression the double ampersand (&&) symbol represents the Cartesian Product operation, and in the second expression the double bar (||) symbol represents a new "Cartesian Sum" operation In both expressions the colon ( ) symbol represents a label operation and the "senc" symbol represents an abstraction operation The curly braces ({}) are used as delimiters in ROSE's notation for name values

```
1 0 X && 2 0 Y senc {POINT}

   POINT
((X 1 0) (Y 2 0))

1 5 FLOAT || 10 FIXED senc {NUMBER}

   NUMBER
((FLOAT 1 5))
((FIXED 10))
```

More details on the algebra can be found elsewhere [Har87] For the moment these details are not important, instead what is important is the principle of being able to manipulate entities in a way that is independent of any particular application This is what relational databases can do for relations and in our opinion it is a major reason for the popularity of relational databases

5. Main Memory

The fourth reason for the efficiency of ROSE is stated below

> 4 During a design session the entities of interest to an application are stored in a main memory workspace

The advantage of this reason is obvious What is less obvious is the way the E-R diagram shown in Figure 4 helps a database programmer decide which entities should be stored in the main memory workspace In particular, the diagram is coded so that

i A solid box in the top left corner of an entity's box means that this entity set is a *database* entity set that will exist for longer than a single design session

ii An open box means that this entity set is an *editing* entity set that will exist only while an application is executing

iii No box means that this set is a *virtual* entity set used to make the description of a storage model simpler These sets are connected to other sets that contain data by IS-A relationships

A typical ROSE transaction proceeds as follows (i) at the start of the transaction a *database* entity is marked (checked out) so that other users will know that it is being edited, (ii) *editing* entity sets are created by taking a copy of the marked entity and distributing its data, (iii) during the transaction the user edits and manipulates the data in the *editing* entity sets, (iv) at the end of the transaction a new *database* entity is created out of the new data in the *editing* entity sets, (v) the new entity either replaces the entity marked at the start of the transaction, or it is added to the *database* entity set as a new entity

The important feature of this protocol is that an *editing* entity set contains data that is normally stored in a *database* entity set Hence, it will represent a small fraction of the total data in a database, and it can be hidden from the other users of a database because they can continue to use the *database* entity Therefore, the *editing* entities of an application can be stored in a private main memory cache while that application is executing

Of course, this protocol is only appropriate when a single user wants to modify an entity This is reasonable for small entities, but in large projects a group of people may

want to modify a very large entity In this case step (ii) above can be modified so that a team leader creates new *database* entities The team members can then modify those entities in the normal way Obviously, this will be harder to manage in practice than it is in principle, but because of the next feature of ROSE described below we can use a code management system (or similar system [Kat87]) to reduce the complexity of this problem

6. Size

ROSE is implemented as a threaded code interpreter for a main memory workspace It allows the entity algebra to be applied to entities stored in this workspace Currently, ROSE contains about 30K lines of C source code that produce about 500K bytes of executable code on a SUN 2/50 This leads to our final reason for the speed of ROSE

5 Small code size allows more space to be used to cache entities

The cause of this reason may be controversial ROSE is small because it uses operating system services to implement database system services For example, entity security is implemented using file system security, entity distribution is implemented using file system distribution, and entity concurrency control is managed using file system concurrency control

ROSE is able to use operating system tools to implement database system services because it stores a *database* entity as a file in an operating system directory This in turn is possible because the typical *database* entity is a relatively large piece of data Therefore, the overhead of storing this entity in its own file is much less significant than the overhead of storing a traditional relational database tuple in its own file

A ROSE database is organized so that an entity set is represented by an operating system directory, an entity is represented by a file in that directory, and an entity's key corresponds to the file name of the file used to stored that entity Hence, in the database of Figure 4 there will be one file in a directory called Interface for each entity in the INTERFACE entity set, and there will be one file in a directory called Logic for each entity in the LOGIC entity set One of the files in the LOGIC directory might be called "4and" and contain the value shown in Figure 3

The critic of this architecture must ask what are the reasons for storing database data in the traditional way [Sto81] We think the only major service missing from ROSE is an associative search engine The entity algebra can be used to program searches, but like the relational algebra it is not as well suited to this task as a language such as SQL Of course, the tools in todays operating systems are not perfect [Rom87] But, CAD/CAM data is different to data processing data, and we think there is more to be gained by exploiting what is available in todays operating systems and what will be available in the future, than by implementing CAD/CAM databases in the traditional way

7 Outline of an Application

In the last five sections we have explained how ROSE has been made fast In this section we will attempt to put these optimizations into context by outlining a typical ROSE application

The first step is to define a database using the techniques shown in Figures 2 and 4 The second step is to define the semantics of that database by defining semantic functions [Har87] These semantics will be common to every application that uses the database and they will encapsulate formulas that show how properties can be calculated using the values stored in entities For example, how the length of a wire can be calculated from the X and Y coordinates of its end points, or how the output of a circuit can be calculated from its inputs

The third step is to write an application that uses the database From the perspective of the application programmer there are three advantages to using ROSE at this step

i Persistence The data in a database will continue to exist after an application has finished execution

ii Sharing Different applications can share the same database

iii Documentation The ROSE data models make the organization of a database easier to understand

A ROSE application can be coded in two ways The first way is to call ROSE from a program written in a conventional programming language The second way is to write programs in ROSE's internal programming language This language allows menu driven, graphics oriented applications to be rapidly prototyped

For example, Figure 5 contains a screen dump of an application that has been written using ROSE's internal language This application allows the user to edit a circuit belonging to the database of Figures 2 and 4 For example, a new wire can be added to a circuit by selecting the "ADD WIRES" option from the menu shown in the top half of the figure When this option is selected a ROSE procedure is called that (i) asks the user to identify two wire pins using a mouse pointer, (ii) draws a line on the screen representing the wire, and (iii) adds an entity to the WIRE entity set of Figure 4 Each line of the procedure is coded as a statement (containing entity algebra expressions) that either sends data to a graphics package or updates an entity set

8. Performance Statistics

To verify that ROSE has good performance we have tested it using an "artificial" application In this application we created an entity set where each entity was a collection of integers The experiments measured how much time was needed to retrieve different numbers of entities in different size entity sets with different size collections We used two measuring instruments a wall clock, and a timing function provided by the VAX/VMS operating system Our measurements were made on a DEC Micro-VAX GPX workstation with 12 Megabytes of real memory The times shown are in seconds

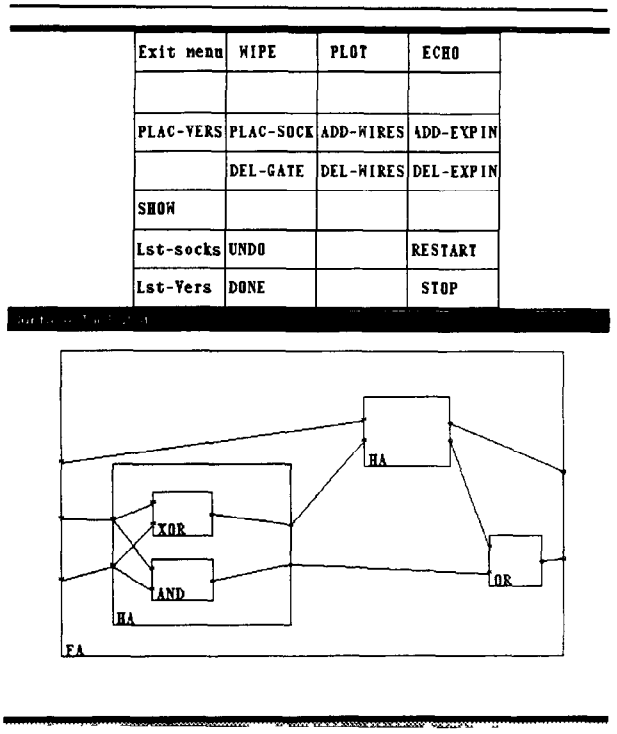

Figure 5 Screen Dump of an Executing Application

Table I shows the effect of varying the size of an entity set while keeping the size of each entity and the number of entities retrieved (the search size) constant Tables II and III vary the other two parameters The following procedure was used to get the numbers shown in these tables

i An entity containing X integers was created where the value of X is given as the Collection Size parameter in each table

ii An *editing* entity set was created containing the entity created at step (i) duplicated Y times where the value of Y is given as the Set Size parameter in each table The first entity in this set was given the key "1", the next entity the key "2" and so on up to "Y"

iii Z keys were created by generating integers between 1 and Z These keys were stored in a variable The value of Z is given as the Search Size parameter in each table

iv The Z key values were used to retrieve Z entities from the *editing* entity set and the time was recorded in a table under the "Unoptimized Time" column

v To measure the effect of our third optimization, step (iv) was repeated and the new time was recorded in a table under the "Optimized Time" column

ROSE organizes the keys of an entity set into an AVL tree [Gon83] The internal AVL keys for this tree are generated by creating a four byte word out of the first character and the last three characters of the entity's key For example, the AVL key for an entity with the key "13690" is "1690" If two or more entity keys have the same AVL key then the addresses of these entities are stored in a linked list The "Unoptimized Time" column of each table measures the time needed by ROSE to search the AVL tree and these linked lists The "Optimized Time" column measures the time needed by ROSE to discover that the address of an entity is already stored within a search key

From the tables we can see that the wall clock times agreed with the VAX system function times until we created entity sets containing more than 300,000 integers For these entity sets the real time performance of the system rapidly deteriorated because of paging problems

The times shown in the tables are fast because they are measuring trivial operations (an AVL search for the unoptimized time, and a comparison for the optimized time) They are uninteresting in the sense that these results could have been predicted However, they are interesting in the sense that these operations are precisely the operations that ROSE needs to perform to retrieve an entity Therefore, they dramatize the effect of the five optimizations described in this paper

In another experiment we determined that 40 entities containing 400 integers each could be stored in 0 25

Table I Varying Set Size

Collection size 10, Search size 10,000				
Size	Unoptimized Time		Optimized Time	
	Wall	Function	Wall	Function
10,000	11 5	10 66	0 9	0 58
20,000	11 5	10 07	0 9	0 58
30,000	12 0	10 06	0 9	0 57
40,000	23 8	11 45	0 9	0 58
50,000	29 5	11 46	0 9	0 58

Table II Varying Collection Size

Set size 10,000, Search size 10,000				
Size	Unoptimized Time		Optimized Time	
	Wall	Function	Wall	Function
20	13 0	12 45	0 9	0 58
30	16 0	13 96	0 9	0 57
40	18 1	15 26	0 9	0 58
50	75 0	18 45	0 9	0 58

Table III Varying Search Size

Set size 10,000, Collection size 10				
Size	Unoptimized Time		Optimized Time	
	Wall	Function	Wall	Function
2,000	1 9	1 91	0 4	0 12
4,000	4 5	3 64	0 5	0 23
6,000	7 2	6 26	0 6	0 35
8,000	9 7	8 62	0 7	0 45

megabytes, a result we expected from theoretical analysis This corresponds to an overhead of three words for each word used to store an integer For richer data structures this overhead will be larger, but we do not expect any data structure to be rich enough to double it to six words Also, in this experiment we were able to neglect the overhead needed to store the key of each entity because this overhead was only twelve words per entity For the larger entity sets shown in the tables this overhead was significant because each key had to be generated and stored twice once for an entity stored in the *editing* entity set, and once for the variable used to perform the searches

9 Conclusion

ROSE is an experimental database system for CAD/CAM applications that has extended and modified relational concepts to make them more suited to CAD/CAM ROSE models databases using the Entity-Relationship model It describes the data structure of an entity using AND/OR trees, and it stores an entity as a file in an operating system directory In this paper we have described five optimizations that have helped to make ROSE a fast database system for CAD/CAM applications

Acknowledgements: This work was partially supported by the Industrial Associates Program of the Rensselaer Polytechnic Institute Center for Interactive Computer Graphics Any opinions expressed or implied are those of the author George Samaras helped me generate the statistics shown in the tables, and design the database described in Figures 2, 3, 4 and 5

References

[Afs85] Afsarmanesh, H, D McLeod, D Knapp, and A Parker, "An Extensible Object-Oriented Approach to Databases for VLSI/CAD", Proc VLDB, Stockholm, 1985

[Bat85] Batory, D S and W Kim, "Modeling concepts for VLSI objects", ACM TODS, 10, 3 (1985), 289-321

[Che76] Chen, P P, "The entity-relationship model-toward a unified model of data", ACM TODS, 1, 1 (1976), 9-35

[Dad86] Dadam, P , et al , "A DBMS prototype to support extended NF relations an integrated view of flat tables and hierarchies", Proc SIGMOD 86, ACM, 1986

[Dit86] Dittrich, K , "Object oriented database systems the notion and the issues", Proc 1986 Workshop on Object Oriented Database Systems, IEEE Computer Society, 1986

[Gon83] Gonnet, G , "Balancing Binary Trees by Internal Path Reduction", CACM, 26, 12 (1983), 1074-1081

[Har87] Hardwick, M , G Samaras and D Spooner, "Evaluating recursive queries in CAD using an extended projection function", Proc Third Data Engineering Conference, IEEE Computer Society, 1987

[Kat85] Katz, R H , *Information Management for Engineering Design,* Surveys in Computer Science, Springer-Verlag, 1985

[Kat87] Katz, R H et al , "Design Version Management", IEEE Design and Test of Computers, February 1987

[Lor83] Lorie, R and W Plouffe, "Complex Objects and Their Use in Design Transactions", Proc Database Week, SIGMOD, May 1983

[Mcl83] McLeod, D , et al , "An Approach to Information Management for CAD/VLSI Applications", Proc ACM Database Week, SIGMOD Conf , San Jose, CA , May 1983

[Rom87] Roman, G , "Data Engineering in Software Development Environments", Proc Third Data Engineering Conference, IEEE Computer Society, 1987

[Sid80] Sidel, W , "Weaknesses of commercial database management systems in engineering applications", Proc Seventeenth Design Automation Conference, IEEE Computer Society (1980), Order No 302, 57-61

[Smi77] Smith, J and D Smith, "Data Abstractions Aggregation and Generalization", ACM Transactions on Database Systems, Volume 3, Number 3, 1977

[Spo86] Spooner, D , M Milicia, and D Faatz, "Modeling Mechanical CAD Data with Data Abstraction and Object-Oriented Techniques", Proc 2nd Data Engineering Conference, Feb, 1986

[Sto81] Stonebraker, M , "Operating System Support for Database Management" Communications of the ACM, Vol 24, No 7, 1981

[Sto83] Stonebraker, M , B Rubenstein, and A Guttman, "Application of Abstract Data Types and Abstract Indices to CAD Data Bases", Proc ACM Database Week, SIGMOD Conf , San Jose, CA , May 1983, 107-113

[Wil85] Wilkins, M W , R Berlin, T Payne and G Wiederhold, "Relational and Entity-Relationship model databases and specialized design files in VLSI design" Proc 22nd Design Automation Conference, IEEE, 1985