The HyperModel Benchmark

Arne J. Berre, T. Lougenia Anderson, Moira Mallison

Oregon Graduate Institute
Department of Computer Science and Engineering
20000 NW Walker Rd
Beaverton, OR 97006-1999 USA

Technical Report No. CSE 88-031

The HyperModel Benchmark

Arne J. Berre
T. Lougenia Anderson
Moira Mallison

Oregon Graduate Center
Department of Computer Science and Engineering
19600 NW Von Neumann Dr.
Beaverton, OR 97006-1999

Technical Report No. CS/E- 88-031

Abstract

This report describes an application-oriented approach to database-evaluation. A generic HyperModel with requirements for database-support similar to many engineering design applications is suggested. The HyperModel is described as schema and operations at a conceptual level, suitable for transformation to different actual database management systems.

The HyperModel Benchmark

Arne J. Berre
Center for Industrial Research
P.O.Box 350, 0314 OSLO 3, NORWAY
uucp: uunet!mcvax!si.uninett!berre

T. Lougenia Anderson
Servio Logic Corp., Beaverton, Oregon, USA

Moira Mallison
Tektronix Inc., Portland, Oregon, USA

Abstract

This report describes an application-oriented approach to database-evaluation. A generic HyperModel with requirements for database-support similar to many engineering design applications is suggested. The HyperModel is described as schema and operations at a conceptual level, suitable for transformation to different actual database management systems.

1. Introduction

Much of the recent effort in database-system development has been aimed at meeting the needs of engineering applications like CASE and CAD. The goal of our work has been to find an approach which makes it possible to evaluate and compare some of these efforts in order to identify database-systems suited for engineering design applications.

This report outlines evaluation-criteria and benchmark-operations for database-support for a generic hypertext model, the HyperModel. The requirements for database-support found for this application are quite similar to requirements stated for other kinds of Engineering applications /BERN87/, and insights gained from this benchmark will therefore also be valuable for a more general set of Engineering applications. For example the recent development of Software Engineering Environments and Integrated Project Support Environments, /GALL86/, has shown that the overall information model for some design-environments has a strong association with the node-link concept in hypertext-models.

The report states the most important requirements for database-support for design-applications. The HyperModel benchmark is then developed as an extension to the database-benchmark presented in /RUBE87/.

2. An application-oriented approach

The best evaluation and benchmarks for database-support for a specific application can only be given by implementing the application for the different database-systems. This will almost always turn out to be too expensive, so the next question is if it is possible to find one generic application which represents common requirements for a larger set of applications.

Things to consider when deciding on database-support for an application include both evaluation of the usability of the system by it's functionality, performance-benchmarks and economic/market-factor analysis.

From the application builder's point of view, performance is not the only critical factor. Functionality is equally important, which suggest that an evaluation should consider more than just performance. The problem for existing databases so far, however, has been that they often don't give adequate performance at all, and then functionality doesn't matter either.

As a generic application for evaluating different database-systems aimed at engineering applications, a slightly extended hypertext model is suggested here. Examples of hypertext-systems for design-environments are: Neptune at Tektronix /DELI86A/, Tender/One-Literate-Programming-System at SI (Center for Industrial Research, Norway) /REEN86/, and the System Factory project at UCLA, /GARG87/. The extension to hypertext is support for an aggregation-relationship through a component/part-of hierarchy. This is a relationship common in many design-applications, and has recently been suggested as a useful extension also to hypertext-systems, /HYPE87/.

The actual database-example will allow creation and editing of the data-part of hypertext-documents. The realization of the display-part is not part of the benchmark-evaluation, but the database will allow for retrieval of the information necessary for an interactive editor, or a formatting program.

3. Requirements for database-support - Evaluation-criteria

Some of the most important requirements for database-support for some hypertext-applications are identified here.

The requirements for database-support can be divided into two categories, requirements for the datamodel and requirements for the system and architecture. The general data-model requirements are derived from the need to support a node-link structure where the nodes may have various types of contents, and links may be created between any nodes. The system and architecture requirements derives from the need to give efficient support in a distributed workstation environment.

The 12 requirements presented here are numbered R1 to R12, R1 to R5 are requirements to the datamodel while R6 to R12 are requirements to the database-system.

3.1. Datamodel requirements

The datamodel should meet the following requirements:

Modeling of complex object-structures (R1)

It should be possible to describe aggregate-structures and arbitrary relationships between components. In a relationship going to many components, the maintenance of an ordered relationship (sequence) should be possible. This is necessary, for instance, to capture a sequence of sections in a document. Both 1-1, 1-N and N-M relationships should be representable, with a possibility of having attributes attached. Recursive

relationships like a hierarchy of sections and sub-sections should also be captured. There should be a natural way to express nodes, links and attributes, and the system should be able to support effificent access and update to them.

In the HyperModel, both an aggregation-relationship to describe a hierarchical document structure, and arbitrary association-links between the nodes should be supported.

Description of different data-types (R2)

The datamodel should be extensible to capture data-types like text and bitmap, and the adding of user-defined types like drawing, picture, voice etc.

The basic types in the HyperModel are TextNode and FormNode (BitMap).

Integration with application programming languages (R3)

There should ideally be no impedance mismatch, /MAIE84/, between an application-programming-language, and the language the programmer uses to make operations on shared and persistent data. Many engineering applications will be written in C or an object-oriented language like C++ or Smalltalk-80. If an editor is written in one of these languages, the operations against the database should not be an unnatural burden on the application-programmer. It should be possible to map generalization- and specialization-relationships from the programming language to the datamodel.

The assumed programming language for the HyperModel editor-application would be an object-oriented language able to represent generalization/ specialization relationships between the types Node, TextNode and FormNode.

Dynamic modifications to the database schema (R4)

It should be possible to dynamically add new types, and specialize existing ones by adding new attributes.

For the HyperModel it should be possible to add a new node-type, DrawNode, e.g. consisting of circles, rectangles and ellipses.

Support for Versions and Variants (R5)

The model should support handling of temporal data. It should be possible to associate a time with all changes to objects, so that a snapshot can be created for any time-point. It should be possible to model parallel versions (variants) of the same object, and to support configuration-structures.

For the HyperModel it should be possible to retrieve the previous version of a node, or a node-structure as it was at a specific time-point.

3.2. Database System requirements

The database-system should meet the following requirements:

An architecture of workstations and servers (R6)

Typically, most engineering applications are intended for a workstation environment. There is at least three different architectural approaches in proposed systems.

- DBMS is local - on workstation (single/multi user systems)
- DBMS is remote - on server (Distributed or central disks)
- DBMS is distributed to workstations - remote/local disks

The essential difference is between single- and multi-user systems, and the extent that different systems support coordination and collaboration between users. Most multi-user mechanisms requires some centralized control which degrades performance. On the other hand single user systems can be partly integrated into the same virtual memory space as the application. Solutions with shared memory for multiple processes have also been used.

For the HyperModel, the architecture should support multiple users in a network of workstations and servers.

Performance suitable for interactive design-applications (R7)

The users will not accept a performance which is significantly lower than what they get when running the applications in single-user mode on their own workstations.

Most hypertext-systems are made for workstations in local area networks, The database-architecture needs to support this efficiently, either by a powerul server or by being distributed on the workstations. There is a tradeoff between letting the database do work remotely, and the need for having fast access to data from an application on the workstation.

An typical application will need access to something between 100 - 10,000 objects per second, where each object is on average 100 bytes in size. This could mean that parts of the database have to be cached/checked-out to main memory in the workstations /ECKL87/.

Concurrency Control (R8)

Short operations on the database should be administrated by a transaction-management mechanism, garantuing consistency in update-/creation-operations.

Cooperation between users (R9)

Long transactions should support cooperation, as opposed to competition, between users. The users will do collaborative work on shared structures, and the system should give support for their coordination. A notion of private and shared workspaces is desirable.

In the HyperModel it should be possible for two users to update different nodes in the

same structure. When one user decides to make his updates shareable, they should be easily accessible for other users.

Logging, backup and recovery (R10)

There should be facilities for doing recovery after crashes on both server-machines and workstations. Garbage collection of non-referenced objects should also be supported.

Access-control (R11)

There should be a mechanism to impose different kinds of access-control policies, for access to shared and persistent data.

In the HyperModel it should be possible to set public read-access for one document-structure, and public write-access for another document-structure. It should still be possible to have links between these structures.

Ad-Hoc Query Language (R12)

In most hypertext-systems so far, a browsing-capability has proved sufficient. As the amount of data grows, however, there might be a need for ad-hoc queries to find a set of nodes satisfying certain criteria.

These 12 requirements are along the lines reported elsewhere /BERN87/, /MAIE84/ and /EURA87/, for other kinds of engineering applications. This leads to the assumption that knowledge about how a database-management-system meets these requirements for a hypertext-system will give valuable information about how it will meet the requirements for a larger set of engineering applications.

4. Existing benchmark-designs

One question to ask is whether we could get some of the knowledge needed to do an

evaluation of actual database-systems from some of the published database-benchmarks.

1) The Wisconsin std. relational benchmark /BITT83/
2) The TP1 - Transaction throughput benchmark /ANON85/
   (Also called the Debit-Credit benchmark)
3) The simple db-operations benchmark /RUBE87/

Unfortunatly none of the most well-known dbms-benchmarks 1) - 3) above seems to provide data with which to evaluate the listed requirements.

For relational databases, The Wisconsin benchmark, /BITT83/, has become the standard. It is essentially a benchmark for query processing, which is aimed at selection-join query-processing, and not the higher level conceptual operations commonly found in engineering applications.

A later benchmark-scheme /ANON85/ concentrated on monitoring system throughput, but this is for a high number of small transactions opposed to a typically small number of long transactions in engineering applications.

In "Benchmarking simple database-operations", /RUBE87/, it is argued that the Wisconsin-test and TP1-test is not suited for benchmarking of databases aimed at engineering-applications A new scheme is proposed, aimed at testing response-time on the following simple operations:

1) Name Lookup
2) Range Lookup
3) Group Lookup
4) Reference Lookup
5) Record Insert
6) Sequential Scan

7) Database open

The example database-model consists of Documents and Persons with a many-to-many relationship between them. It is especially aimed at simple db-operations for engineering- applications, but the datamodel is too simple to measure transitive closures and other traversal operations which are common in engineering-applications.

The described operations can give useful information on database-performance, but many database-system will be able to support some higher level conceptual operations more efficiently than others, so knowledge about simple operations only might not be enough. For instance could a notion of complex objects based on an aggregation-relationship allow for clustering of data, which would make transitive closure operations perform more efficiently. Also the Person-Document model does not show the problem of collaboration between multiple users editing parts of the same data-structure.

We found this benchmark interesting, but would like to see it in the context of a more typical engineering application. Hence the HyperModel benchmark incorporates the same 7 operations, but uses an example model with a more complex structure, and some additional operations appropriate for this structure.

5. The HyperModel Benchmark

This section describes the conceptual schema to be used in the HyperModel benchmark. The goal has been to make a high-level description which can be mapped into a realization on different database-systems. For this purpose we have chosen to represent the shema using the Object Modeling Technique (OMT) described in /BLAH88/ and /LOOM87/. OMT is based on the LRDM-extension /TEOR86/ to the ERA-model /CHEN76/ and allows Generalization/Specialization and Aggregation. Since ordered relationships is something which is required for the HyperModel, a notation for representing those is added here.

5.1. The Conceptual Schema

A HyperModel-document consists of a recursive structure of sections. The sections on the same level have an internal ordering. The context of a section can be either text or a form (bitMap). References or hypertext-links can be made between any two nodes. Attributes attached to the link allows for a description of link-offset within the nodes.

The recursive structure is described by a 1-N aggregation-relationship parent/children, and a M-N aggregation-relationship partOf/parts between nodes. The links are described by an M-N association-relationship refTo/refFrom with two attributes, offsetFrom/To. The fact that some nodes contains text or a form (bitMap) is described by a generalization/specialization-relationship between the Node-class and the TextNode- and FormNode-classes.

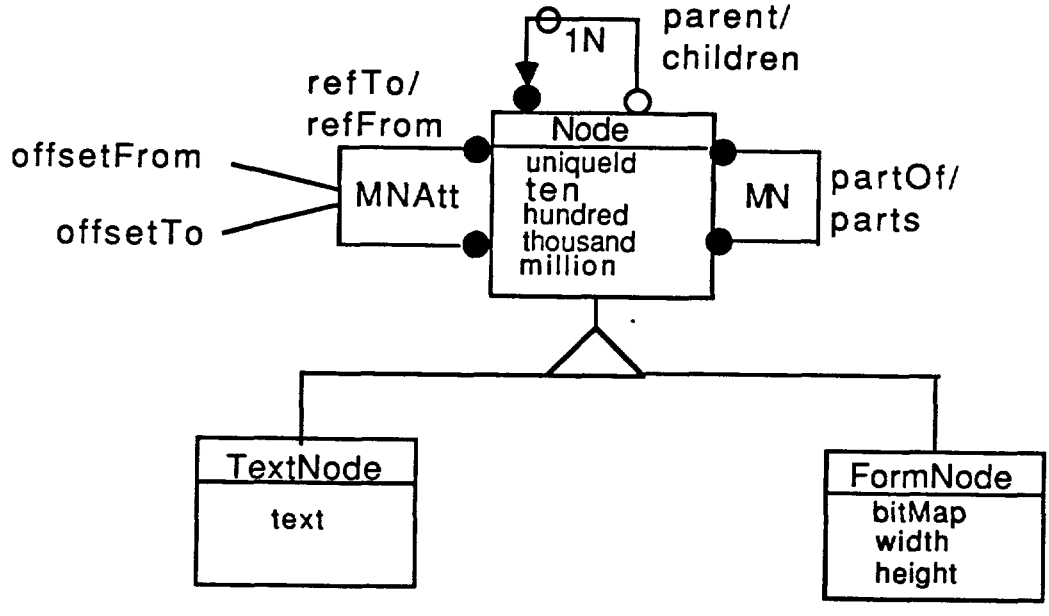

Figure 1 - The HyperModel schema

In Figure 1, lines represent bidirectional relationships, black circles represent a many-end of a relationship, while white circles represent a one-end. The arrow points towards the composite object in an aggregation-relationship. A circle on the relationship-line

means that this relationship should be ordered. Attributes are described inside a class-object. The triangle on the lines between the classes symbolize generalization.

The ten-, hundred-, thousand and million-attributes should be randomly assigned to integers in the corresponding interval. The uniqueId attribute should be unique for each node, for instance numbering the nodes.

Each text-node contains a text-string of a random number (10-100) of words, the words separated by a space and consisting of a random number (1-10) of random small characters The first, middle and last word should be "version1".

Each form-node should initially be white (all 0's), with a bitmap-size varying randomly between 100x100 and 400x400.

5.2. Test-database generation

The test-databases consists of a network-structure of nodes, describing two aggregation-relationship-types, 1-N and M-N, and one association-relationship-type, M-N with attributes, between the nodes. One of the aggregation-relationship-types describes a hierarchical 1-N tree-structure, while the other describes a hierarchical M-N structure with shared sub-parts.

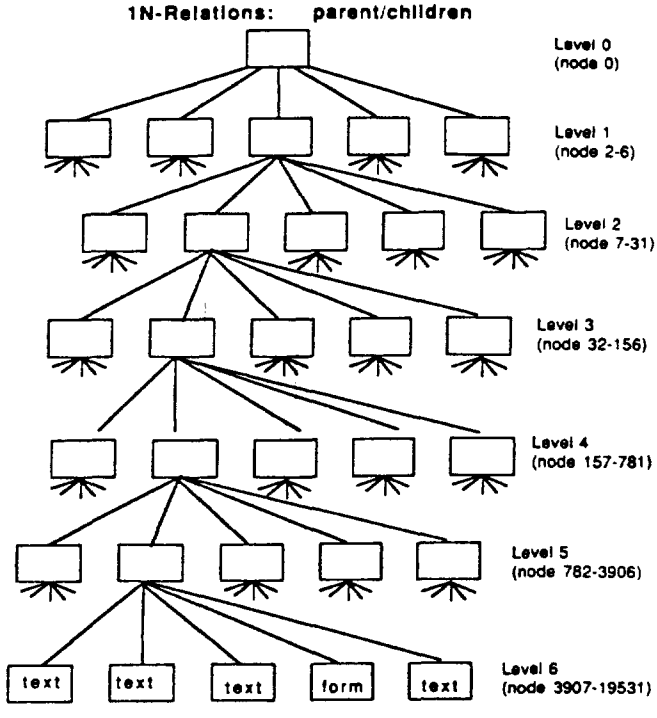

Figure 2 - The 1-N Relationship

There should be test-databases of 3 different sizes, created with leaf nodes on level 4, 5 and 6 in the 1-N aggregation hierarchy. The test-databases have a fan-out of 5 new nodes for each level in the 1-N hierarchy. This gives the following number of nodes on each level: 0(1), 1(5), 2(25), 3(125), 4(625), 5(3125), 6(15625), and a total of 19531 nodes for level 6, adding one level will give a total of 97656 nodes. The leaf-level consist of text-nodes and form-nodes. There is one form-node per 125 text-nodes, giving 125 form-nodes and 15500 text-nodes in the level database. One semantic interpretation of the 1-N aggregation hierarchy with leaf nodes on level 6 could be an archive with 5 folders with 5 documents in each folder. Each document will contain 5 chapters with 5 sections with 5 subsections with 5 text or bit-map nodes.

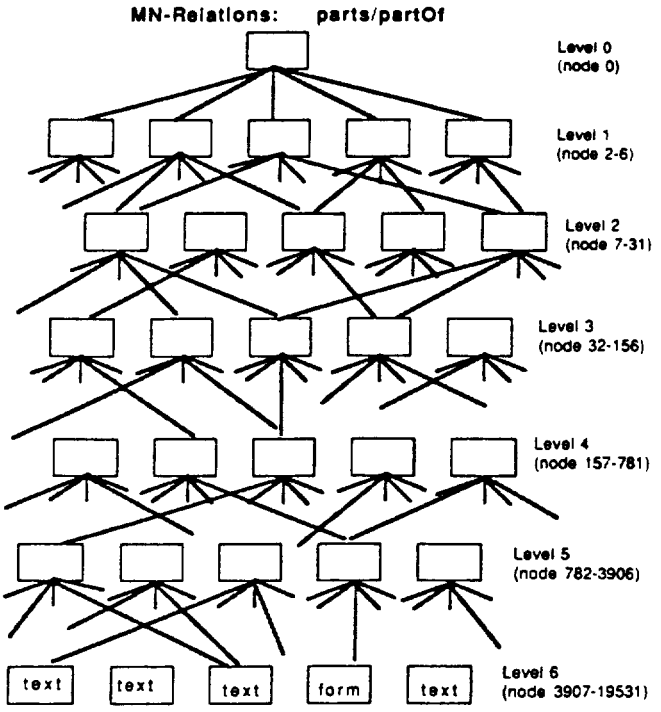

Figure 3 - The M-N Relationship

The M-N is created for each node, by relating it to 5 random nodes from the next level

in the 1-N aggregation-hierarchy. Do this for all nodes, except the leaf nodes. This gives a number of 1-N and M-N relationships one less than the number of nodes.

This also gives the possibility to measure the effect of an eventual clustering along one of the aggregation hierarchies. If the system supports clustering, clustering should be done along the 1-N relationship-hierarchy.

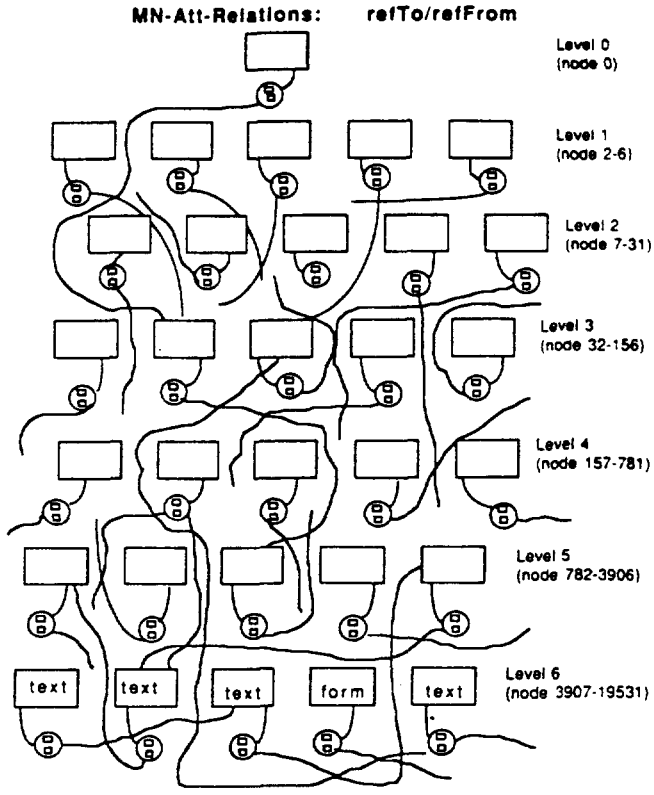

Figure 4 - The M-N Relationship with Attributes

The M-N relationship with attributes gives a possibility to create a directed weigthed graph. The offsetTo/offsetFrom attributes can be used to keep the weigths in each direction. The refTo/refFrom relationships are created by visiting each node once and creating a reference to another random node. The values of the offset-attributes should be random between 0 and 9. This will give a number of M-N attribute relationships equal to the number of nodes.

With an approximate size of 80 bytes per Node, 380 bytes per TextNode, 7800 bytes per

FormNode and 25 bytes per link-reference, the total size of the data in the database with 19531 nodes will be around 8 MB. Increasing the number of levels with one will increase the size of the database by 5, to around 40 MB.

N.B. The given number of levels and fanouts, and the sizes of text and form-nodes should not be interpreted as a static picture, which a database-schema could be build around. It should be possible to increase and decrease the number of levels, the fanouts, the size of text and the size of a bitmap in any database. - The schema-definitions and operations should take this into account. The uniqueId-attribute should not be exploited to determine any sequence or position of a node in the database structure. The schema or closure-operations should not encode knowledge that closure-operations start at level three in the structure. The regularity of the schema is made to have a predictable number of nodes involved in operations based on randomly found nodes. The random numbers should be drawn from a Uniform distribution for the actual interval.

5.3. Operations for Database Creation

Time for object and relationship creation is measured when the test databases are created. The measured time is split into time for internal and leaf node creation and time for the creation of the different relationships, with corresponding commit. This time includes the time to create a node-object with the objects necessary for holding relationship-references and the initialization of the three integer-attributes to random values and the attribute holding the identifier value. The time also includes time to update appropriate indexing-structures. The measurements should be done for each of the differents sized test-databases.

This gives the following operations to be measured:

a) Create Internal Nodes (milliseconds per node)

commit-time for the operation

(b) Run the operation 50 times. This is called the *cold run.*

(c) Commit the changes to the database.

(d) Repeat the operation again 50 times with the same 50 nodes to test the effect of cacheing. This is called the *warm run.*

(e) Close the database to prevent cacheing from this operation sequence from having an effect on the next operation sequence.

In a workstation/server architecture the cold run would require fetching of nodes from the server.

When input to or output from an operation is to be a node or a set of nodes, it is assumed to be a reference to a node and not a copy of the node itself. In a relational system it would typically be the value of a key attribute, in an object-oriented system it would be an object identifier maintained by the system. A set or list of nodes returned from an operations (as set or list of key-attributes or object identifiers) should itself be storable in the database. In the following it is said "a node" or "a set of nodes", whithout stating how it might be represented.

Time-measurements should be normalized to millisecond per node returned from the operation, and should be measured for both the cold and the warm run on each operations, for each level (4,5,6) of the database. The database should be in a stable state before and after each operation, so datbase-commit-time should be included in the measurement.

In the following we give a brief description of the purpose of each of the operation categories, followed by the specific operations within each category. For each operation, we specify the operation with respect to the HyperModel data structures, the expected input and the expected output.

6.1. Name Lookup

This operation gets the hundred-attribute of a node based on a reference to the node. The reference can either be given by the value of a unique attribute (key), or by a system-generated identifier, an object id. Both kinds of lookup should be measured if applicable.

/ 01 */ nameLookUp();* Specification: Find the Node instance with the uniqueId attribute equal to n and return the value of its hundred attribute. *Input:* A random integer between 1 and the maximum number of nodes. (Random numbers are drawn from a uniform distribution) *Output:* The value of the hundred attribute for the corresponding node.

/ 02 */ nameOIDLookup();* Specification: Find the Node instance with a given object id and return the value of its hundred attribute. Input: An object id (a random object). Output: The value of the hundred attribute for the corresponding node.

6.2. Range Lookup

The operation finds the nodes satisfying a range-predicate based on the values of the hundred- or million-attribute.

The range-lookup has a selectivity of 10% for the hundred-attribute and 1% for the million-attribute, and allows for the use of an indexing-mechanisms on the hundred and million-attributes.

/* 03 */ rangeLookUpHundred(); Specification: Get the set of all nodes that have the hundred attribute in the range x..x+9 (10% selectivity). *Input:* A pair of numbers (x, x+9), where $1 \leq x \leq 90$. *Output:* The set of nodes whose hundred attribute satisifes the predicate.

/* 04 */ rangeLookUpMillion(); Specification: Get the set of all nodes that have the mil-

lion attribute in the range x..x+9999 (1% selectivity). *Input:* A pair of numbers (x, x+9999), where $1 \leq x \leq 990000$. *Output:* The set of nodes whose million attribute satisfies the predicate.

6.3. Group Lookup

These operations follow the 1-N, the M-N, or the M-N Attribute relations from a random node. The average time per node to retrieve a reference to the related nodes, is measured.

For the 1-N and M-N relationships the time should show time per relationship access, and the total time for one operation will be five times this, since five relationships will be found. For the M-N Attribute relation, only one relationship will be found.

/ 05A */ groupLookUp1N();* Specification: Get the children nodes of a random node. *Input:* A random internal node *Output:* An ordered list (sequence) of the five children nodes of the given node. (This might be more efficient than the next groupLookUpMN since clustering may be used along the 1-N hierarchy).

/ 05B */ groupLookUpMN();* Specification: Get the part nodes of a random node. *Input:* A random internal node. *Output:* A set of the five nodes beeing the parts of the given node.

/ 06 */ groupLookUpMNATT();* Specification: Get the node related to a given node by the M-N attribute relation refsTo. *Input:* A random node *Output:* A set containing the node referenced by the given node.

6.4. Reference Lookup

This is the inverse of group-lookup, given by following the relationships in the direction opposite to that of groupLookup.

/ 07A */ refLookUp1N();* Specification: Given an node, find its parent node. *Input:* A random node, except the root-node . *Output:* A set containing the parent node of the given node.

/ 07B */ refLookUpMN();* Specification: Given an node, find the node(s) it is part of. *Input:* A random node, except the root-node. *Output:* A set containing the node(s) this node is part of.

/ 08 */ refLookUpMNATT();* Specification: Get the nodes related to a given node by the M-N attribute relation refsFrom. *Input:* A random node *Output:* A set (possibly empty) containing the nodes that reference the given node.

6.4.1. Sequential Scan

Find the average time per node when all nodes of the test-structure are visited. This should be done by accessing the ten-attribute of each.

The database should be allowed to have other instances of class Node, (eg a second copy of the test-database, so the direct extension allInstances-of-Node cannot be used. A requirement is that the application-program can simultaneously access other node-objects not in the test-database-structure

/ 09 */ seqScan();* Specification: Get the ten attribute of each node in the test database, without performing any particular operation on the attribute. *Input:* None. *Output:* None. - Note that the ten-attribute was retreived for each node, ensuring node access, but no result was actually returned. In systems supporting access to one object at a time, the ten-attribute would be retrived and assigned to a variable for each node sequentially, instead of returning all ten-attribute-values at once.

6.5. Closure Traversals

These operations will start with a random node, typically on level three, and perform the operations on the nodes transitively reachable by a certain relationship from this node. For all the operations, measure the average time per node involved in the operation.

The closure-operations is required to preserve the order of the sub-node relationships where appropriate (for 1-N), and deliver a list of references according to a preOrder traversal of the structure. The list should be storable in the database. This could for instance be usable in a simple table of content for the structure. The M-N Attribute relationship will never reach a node without an outgoing relationship, (i.e., no terminating condition exists) and is specified to be traversed to a depth given at run-time, here 25. The two other relationships should be traversed down to the leaves.

Since eventual clustering should be used for the 1-N relationship it is assumed that this M-N will take longer time than closure1NOID, when not cached.

The timing of the closure-operations is the average time per node found. The total time will be n times larger than this, where n-level4= 6, n-level5= 31 and n-level6= 156.

/ 10 */ closure1N();* Specification: Get a list of all nodes reachable from a random node on level three, by following the 1-N relationship recursively to the leaves of the tree. The list should be ordered according to a pre-order traversal of the structure. *Input:* A random node on level 3 (31-156) *Output:* A list of all nodes reachable from the given node by following the 1-N relationship recursively.

/ 14 */ closureMN();* Specification: Get a list of all nodes reachable from a random node on level three by following the M-N relationship recursively to the leaves of the tree. Since eventual clustering should be used for the 1-N relationship it is assumed that this may take longer than closure1N. *Input:* A random node on level 3 (31-156), *Output:* A list of all nodes reachable from the given node by following the M-N relationship recur-

sively.

/ 15 */ closureMNATT();* Specification: Get all nodes reachable from a random node on level three by following the M-N attribute relationship recursively to a depth given at run-time, here twenty-five. *Input:* A random node on level 3 (31-156), *Output:* A list of all nodes reachable from the given node by following the M-N attribute relationship recursively to a depth of twenty-five.

6.6. Other closure operations

In addition to closure-traversals, four other operations on transitively reachable nodes are specified. There is one operation to sum the value of the hundred-attributes, one operation to set the value of this attribute, one operation to return all nodes satisfying a range-predicate and one operation to get the distance to nodes measured by the offsetFrom-attribute of the many-to-many relationship with attributes.

/ 11 */ closure1NAttSum();* Specification: Get the sum of the attribute of all nodes reachable from a random node on level 3 by following the 1-N relationship recursively to the leaves of the tree. *Input:* A random node on level 3 (31-156), *Output:* An integer representing the sum of the attribute for all the nodes visited.

/ 12 */ closure1NAttSet();* Specification: For all nodes reachable from a random node on level three by following the 1-N relationship recursively to the leaves of the tree, set the attribute to ninety-nine minus the current value. (By doing this twice the attribute is restored to its original value.) *Input:* A random node on level 3 (31-156), *Output:* None. The nodes in the database will have been updated, however.

/ 13 */ closure1NPred();* Specification: Get all nodes reachable from a random node on level three by following the 1-N relationship recursively to the leaves of the tree, and excluding nodes as well as terminating recursion at the nodes that have the million

attribute in the range from x to x+9999. *Input:* A random node on level 3 (31-156), *Output:* A set of nodes that are reachable from the given node and that satisfy the predicate.

/ 18 */ closureMNATTLINKSUM();* Specification: Get the total distance, measured by the offsetTo attribute, of all nodes reachable from a random node on level three by following the M-N attribute relationship recursively to a depth given at run-time, here 25. Return a list of pairs containing the node and associated distance. *Input:* A random node on level 3 (31-156), *Output:* A set of pairs whose first elements are the node that are reachable from the given node and whose second element is the distance to this node from the initial node, measured by the sum of the offsetTo attributes along the path (i.e., the M-N attribute relationship).

6.7. Editing

These operations will show how powerful the database-programming language is, and if any statements have to be executed in another programming language. The purpose of these operations is also to check the overhead it takes to update a node, assuming that this node has just been updated by someone else. For a textNode, the node has to be checked-out or ensured write-access, then a substring is substituted and the node is released for others to update.

The TextNodeEditOperation gets a random textNode and substitute the occurence of substring "version1" with "version-2" in the first run, then substitute back again in the second run (Note that "version-2" will be one character longer). The measured time shows the average time of one edit-operation, and includes time to retrieve and store the node.

/ 16 */ textNodeEdit();* Specification: Get a random text node and substitute the occurence of sub-string "version1" with "version-2" in the first run, then substitute back again in the second run. *Input:* A random text-node *Output:* None, however, the node will have been updated.

The FormNodeEdit-operation inverts a subrectangle (25x25,50x50) in a random selected formNode. The measured time shows the average time of one edit-operation, and includes time to retrieve and store the node.

/ 17 */ formNodeEdit();* Specification: Invert a subrectangle (25x25,50x50) in a random selected formNode. *Input:* A random form-node, N.B. - The same form node is used for the fifty repetitions of the operation. *Output:* None, however, the node will have been updated.

6.8. Possible Extensions to the Operation Set

There are other operations that might prove useful in asessing support for the listed requirements. These include the following.

(1) Add a new type, or add an attribute to an existing type to assess schema modification capabilities (R4).

(2) Create a new version and find the previous version or a specific version of a node, to assess version handling capabilities (R5).

(3) Set the access to the nodes in a specific document to read-only or non-access for public, to check on access control (R11).

7. Conclusions and Future Work

The HyperModel benchmark is a benchmark specifically designed to test the performance of Object-Oriented DBMSs for engineering applications. However, we feel that it will also be useful to evaluate existing DBMS performance, particularly to test the performance of their programming languages interfaces. Furthermore, the HyperModel is also a good starting point to benchmark OO-DBMSs for applications other than engineering, since it tests two of the most important features of OO-DBMSs, complex object representation and complex operation implementation.

The HyperModel has been implemented for Vbase, Gemstone, and Smalltalk-80, this will be documented in /ANDE89/. It is currently being implemented on a relational system following the methodology outlined in /BLAH88/. There are also plans to implement if on some of the recent European Object Management Systems, like Damokles /DITT86/ and PCTE-OMS /GALL86/

A second aspect to be investigated is the impact of a multi-user environment on the design and results of the benchmark. We have done some experiments with multi-user aspects by starting up two and more HyperModel applications in parallel and running the operations as for the single user case. However, since the systems we have worked with support optimistic concurrency control, it is a problem to define update operations that do not conflict. This is an area for future work in the HyperModel benchmark design. Based on the experiences with the benchmark on different systems, a new version of the benchmark is planned for late 1989. A framework for setting up the database and the queries, written in C, is available from the authors.

8. Acknowledgements

This work was done while the authors where at Tektronix, and during the *Database Year 1987-88* sponsored by the Oregon Graduate Center and the Oregon Center for Advanced Technology Education. A special thank is due to Professor David Maier for arranging the *Database Year* and providing numerous valuable discussions for the benchmark design. Thanks also to Rick Cattell and the other participants at the "Workshop on Object-Oriented Database Benchmarking" in March 1988, arranged by the Oregon Database Forum. Arne J. Berre was visiting Oregon Graduate Center and Tektronix during the *Database Year* , and was supported by the Royal Norwegian Council for Scientific Research.

9. References

/ANDE89/ Andersen, T.L., Berre A., Mallison, M., Porter H., Schneider B.
"The Tektronix HyperModel benchmark" - Benchmark-results for two Object-Oriented DBMSs - Gemstone and Vbase.
Submitted for publication

/ANON83/ Anon & al
"A Measure of Transaction Processing Power"
Datamation 31-7, April 1985

/BERR89/ Berre A.J., Anderson T.L.
"Vbase and the HyperModel benchmark"
To be published in a book in 1989: Horowitz, Ellis.
"Object-Oriented Databases and Applications"

/BERN87/ Bernstein, P
"Database System Support for Software Engineering - An Extended Abstract"
9th International Conf. on Software Engineering, Monterey, April 1987

/BITT83/ D. Bitton, D. DeWitt, C. Turbyfill
"Benchmarking database systems - A systematic approach"
Univ. of Wisconsin-Madison, Tech Report #526, December 1983

/BLAH88/ M. Blaha, W. Premerlani, J. Rumbaugh
"Relational Database Design using an Object-Oriented Methodology" CACM Vol. 31, Nr. 4, April 1988

/CHEN76/ Chen, P.
"The entity-relationship model: Toward a unified view of data"
ACM TODS 1, 1, March 1976

/CONC87/ Conklin, J.
"Hypertext: An Introduction and Survey"
IEEE Computer, September 1987

/DELI86/ Delisle, N., Schwartz, M.
"Neptune: A Hypertext System for CAD Applications"
SIGMOD-86, page 132-141

/DITT86/ Dittrich, K., Gotthard, W., Lockemann, P
"DAMOKLES - A Database System for Software Engineering Environments"
Proc. of IFIP workshop on Advanced Programming Environments, Trondheim, Norway, June 1986

/ECKL87/ Ecklund, D., Ecklund, E.
"CAD Performance Requirements for Persistent Object Systems in a Distributed Environment"
Proceedings of the Workshop on Persistent Object Systems: their design, implementation and use, Appin, August 1987

/EURA87/ Inter-Europe Procurement Group - IEPG TA-13
"Requirements and design criteria for tool support interface"

Requirements to PCTE+, July 1987

/GALL86/ Gallo, F., Minot, R., Thomas, I.
"The Object Management System of PCTE as a Software Engineering Database Management System"
Second ACM Software Engineering Symposium on Practical Software Development Environments,
Palo Alto, December 1986, 12-15

/GARG87/ Garg, P., Scacchi, W.
"On Designing Intelligent Hypertext Systems for Information Management in Software Engineering"
Hypertext'87, page 409

/HYPE87/ Halasz, F.G.
"Reflections on Notecards: Seven Issues for the Next Generation of Hypermedia Systems"
Comm. of ACM, Vol-31, Nr. 7, July 1988

/LOOM87/ M. Loomis, J. Rumbaugh
"An object-modeling technique for conceptual design"
ECOOP-87

/MAIE84/ Maier, D.
"Data Model Requirements for Engineering Applications"
1st Workshop on Expert Database Systems, 1984

/REEN86/ Reenskaug, T.
"The Tender/One Environment"
Technical Report 86-15, Center for Industrial Research
Oslo, Norway

/RUBE87/ Rubenstein, W., Kubicar, M, Cattell, R.
"Benchmarking Simple Database Operations"
SIGMOD-87, page 387-394

/TEOR86/ Teorey, T., Yang, D., Fry, J.
"A logical design methodology for relational databases using the extended entity-relationship model"
ACM Comput. Surv. 18, 2, June 1986