TECHNICAL REPORT NO. 282

A Grammar Model for Databases

by

Latha S. Colby and Dirk VanGucht

June 1989

COMPUTER SCIENCE DEPARTMENT

INDIANA UNIVERSITY

Bloomington, Indiana 47405–4101

A Grammar Model for Databases

Latha S. Colby
Dirk Van Gucht

Indiana University
Department of Computer Science
Bloomington, IN, 47405

Abstract

A model based on grammars in which the schema of a database is represented using production rules and the instances by strings parsed by these rules was originally proposed for modeling text databases. We show, in this paper, that the grammar model can support modeling constructs provided by both traditional and semantic data models and can, in addition, support applications such as hypertext systems that are not easily supported by any of the existing data models. We then show that in addition to being a valuable model in its own right, this model provides a powerful notation for describing other models and can hence be used as a unifying framework for various data models. We also describe some of the storage structures necessary to access and manipulate objects represented in this model. The target of our investigations in the area of grammar models is a generalized database system, with the grammar model as the underlying database model, which would allow data to be viewed and manipulated in any one of several known models.

1. Introduction

Research in data modeling and data manipulation has lead to the creation of more and more sophisticated data models due to the ever increasing complexity of data applications. Although traditional database models like the relational, network and hierarchical models are used in a number of database applications, there are several areas of application for which these models are unsuitable or insufficient. In order to provide a better representation for complex objects the nested relational model [15, 19] was developed as an extension of the traditional (flat) relational model by allowing relations to be composed of other (sub) relations. While this model and other complex object models [1] do overcome some of the limitations of the traditional models, they are still unsuitable for many applications.

There has been substantial interest among database researchers and users in semantic database models, which allow more semantics to be associated with data. These models, like the IFO [2], ER [5], SDM [13], FDM [18], to mention a few, provide better data modeling capabilities with a rich set of constructs like abstract data types, classification, specialization, inheritance, etc., none of which are supported by either traditional database models or the nested relational model. Object-oriented database systems such as, O_2 [3], Gemstone [7], IRIS [9], ORION [16], also provide different degrees of support for concepts like classification, inheritance, etc., although in most of these systems greater emphasis is placed on the procedural aspects of data than on the modeling aspects.

Although there are several semantic and object-oriented models that enable complex interrelationships between data to be easily represented, there are some applications, such as text databases, (*e.g.*, dictionaries, encyclopedias, notecards), for which these models are not entirely suitable. Gonnet and Tompa [10] have proposed a model based on grammars for text databases in which the scheme of the database is represented by a set of production rules and the instances by strings parsed by these rules. We show, in this paper, that the grammar model easily allows for constructs like specialization and generalization in addition to traditional database concepts like aggregation and set formation. We also show that this model is not just another semantic model but that its flexibility and modeling power allow most other data models to be modeled very easily in it. In other words, the grammar model can serve as the underlying basis for a generalized database system that can support other data models and a very wide range of database applications. Our main motivation for choosing the grammar model as the underlying basis for a database system is its simplicity, which allows for fairly simple and straightforward specifications of query languages and storage structures and also for theoretical investigation of data modeling and querying issues. In the next section, we describe the grammar model. In Section 3, we discuss the modeling power of the grammar model and its advantages over other models. In Section 4, we describe certain implementation issues and finally, in Section 5, we outline future research possibilities.

2. The Grammar Model

The main idea behind the grammar model came from the observation that the structure of text data (its schema) can most conveniently be described by a formal grammar, whereas the actual data can be represented as a word together with its parse tree, called a *p–string*, over this grammar. Consider the following example inspired by Gonnet and Tompa [10].

Example 1

Suppose we want to create a reference list consisting of books and journal articles. An example journal article is

> Abiteboul, S., and Hull, R. IFO: A formal semantic database system. *ACM Transactions on Database Systems 12*, 4 (Dec. 1987), 525–565.

and an example book entry is

> Maier, D. *The theory of relational databases*. Computer Science Press, Rockville Md., 1983.

A grammar for such a reference list (where | separates alternatives, * represents zero or more, + represents one or more, parentheses represent grouping, and character strings surrounded by quotes represent terminal symbols) is:

REFERENCE-LIST ⟶ ENTRY*
ENTRY ⟶ BOOK | ARTICLE
BOOK ⟶ AUTHORS BOOK-TITLE
BOOK-SOURCE
ARTICLE ⟶ AUTHORS ARTICLE-TITLE
ARTICLE-SOURCE
AUTHORS ⟶ (AUTHOR ",")* "and"
AUTHOR | AUTHOR
AUTHOR ⟶ SURNAME "," INITIALS
SURNAME ⟶ ROMAN-TEXT
INITIALS ⟶ (CHAR ".")*
BOOK-TITLE ⟶ ITALIC-TEXT "."
BOOK-SOURCE ⟶ PUBLISHER "," LOCATION
"," YEAR "."

PUBLISHER ⟶ ROMAN-TEXT
LOCATION ⟶ CITY STATE
CITY ⟶ ROMAN-TEXT
STATE ⟶ CHAR CHAR "."
YEAR ⟶ NATURAL-NUMBER
ARTICLE-TITLE ⟶ ROMAN-TEXT "."
ARTICLE-SOURCE ⟶ JOURNAL VOLUME "," NUMBER
DATE "," PAGENUMBERS
JOURNAL ⟶ ITALIC-TEXT
VOLUME ⟶ ITALIC-NATURAL-NUMBER
DATE ⟶ "(" MONTH NUMBER ")"
MONTH ⟶ "Jan."| "Feb."| ··· | "Dec."
PAGENUMBERS ⟶ NUMBER "-" NUMBER
NUMBER ⟶ NATURAL-NUMBER

In this example we assume that ROMAN-TEXT (ITALIC-TEXT) is a string of roman (italic) characters and that NATURAL-NUMBER (ITALIC-NATURAL-NUMBER) is a positive number, represented with roman (italic) digits. The data for the reference list example can then be represented by a p–string

over this grammar. Figure 1 shows the part of the p–string corresponding to the book entry.

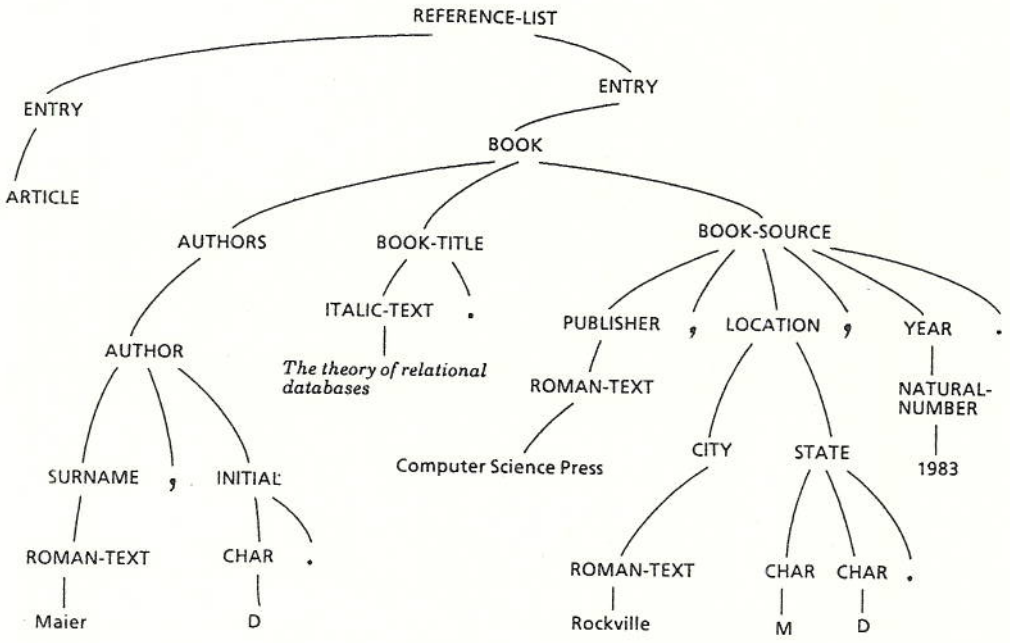

Figure 1. P-string corresponding to the book entry of Example 1.

Although it can be argued for the reference list example that traditional data modeling can capture the relationship between authors, title and source in a reference list entry, in so doing the essence of the entry, i.e. its form, would be lost [10]. This makes for unnatural and often involved data processing. Moreover, traditional data models cannot easily handle the large variety of bibliographical formats that are found in such databases. While Gonnet and Tompa showed the usefulness of the grammar model in the context of text databases, they did not consider the suitability of their model to other database applications. In the next section we show that the grammar model goes beyond being a model for just text-oriented applications. We show that this model can support most of the modeling constructs found in other data models and we also show that the grammar model has several advantages over many other data models.

3. Advantages of the Grammar Model

We first show that the grammar model can not only support most of the semantic database constructs like specialization and generalization, but that it has several advantages over most semantic databases. We then show how many of the traditional data models correspond to families of grammar models defined by restricting the productions to certain formats or templates. Finally, we consider a non-traditional application for databases, i.e., hypertext systems, and show that the grammar model is natural for modeling such applications that are not easily modeled by any of the existing data models.

3.1 Modeling Semantic Databases

It is obvious from the text-database example in the previous section that aggregation and set formation are easily modeled by grammars. Most semantic data models allow specialization and generalization of classes. Generalization can be easily modeled in the grammar model. In Example 1, the production ENTRY $\longrightarrow$ BOOK | ARTICLE is an example of generalization since an instance of ENTRY can be either a BOOK or an ARTICLE. This was first shown by Brodie and Ridjanovic in [4]. A specialization of a class is a (sub)class which has a subset of the instances in the parent class. The instances of the subclass inherit all the attributes (and methods in object-oriented data models) of the parent class. The following example shows how specialization can be represented in the grammar model.

Example 2

Let us suppose that we want a database of persons associated with a university with the attributes NAME, AGE and ADDRESS. The productions given below represent the scheme for such a database.

PERSON-DB $\longrightarrow$ PERSON*
PERSON $\longrightarrow$ NAME AGE ADDRESS
ADDRESS $\longrightarrow$ CHAR-STRING
AGE $\longrightarrow$ NATURAL-NUMBER
NAME $\longrightarrow$ CHAR-STRING

Now, let us suppose that we want to create a separate class for students in the university with the additional attributes MAJOR, STANDING and COURSES-TAKEN. The production rules given below represent the part of the structure that is specific to instances of the STUDENT class.

STUDENT $\longrightarrow$ COURSES-TAKEN MAJOR STANDING
COURSES-TAKEN $\longrightarrow$ COURSE-TAKEN*
COURSE-TAKEN $\longrightarrow$ CNO SEMESTER GRADE
CNO $\longrightarrow$ CHAR-STRING
SEMESTER $\longrightarrow$ CHAR-STRING
GRADE $\longrightarrow$ "A" | "B" | "C" | "D" | "F"
MAJOR $\longrightarrow$ "CSCI" | "MATH"
STANDING $\longrightarrow$ "Fr" | "So" | "Jr" | "Sr" | "Gd"

But since STUDENT is a subtype of PERSON, an instance of the former must also have the attributes

of the latter. This can be done by generating a production rule which is derived from the rule that describes the type PERSON and appending 'STUDENT' to the right hand side of the rule.

PERSON $\longrightarrow$ NAME AGE ADDRESS STUDENT

This rule is then added to all the other rules to get the final scheme. So, we now have two rules describing the PERSON class – one rule representing objects that belong only to the PERSON class and the other representing objects belonging to the STUDENT (and hence to the PERSON class). Figure 2 shows an example of instances represented according to this scheme.

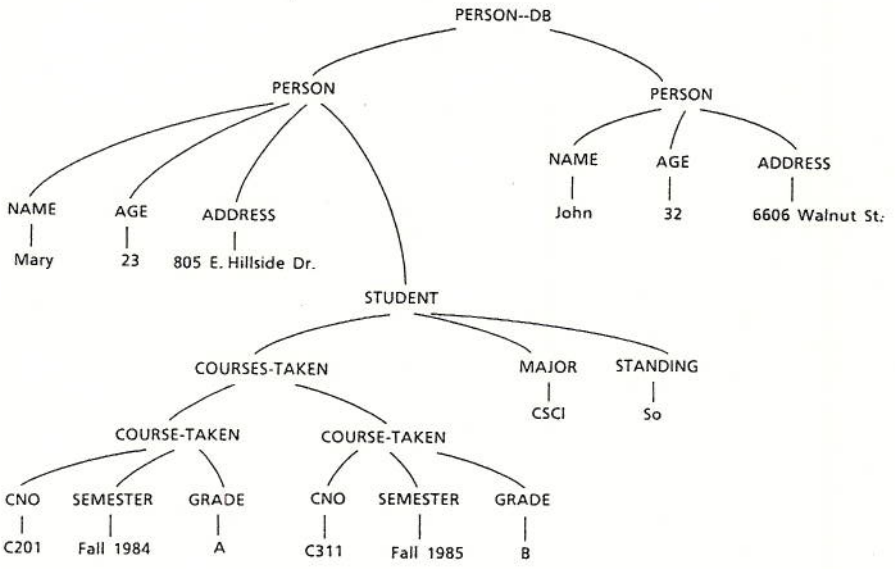

Figure 2. An example of the PERSON-DB

Although most semantic database models support constructs like aggregation and specialization, there are several advantages to representing semantic databases in the grammar model. We list some of the advantages below.

- **Update propagation is handled easily.**
  Let us suppose that we were to remove the instance corresponding to Mary from the PERSON class. In the grammar model we would simply remove the entire subtree with root node PERSON corresponding to Mary without having to worry about deleting the instance of Mary in the subclass STUDENT as well since it is deleted when the PERSON instance is deleted. On the other hand if we were to delete the instance of Mary from the STUDENT class alone, we would only need to remove the subtree with root node STUDENT from the p-string with root PERSON corresponding to Mary. Insertions are also easily handled.

- **Classes can have objects of different data types.**
  Consider the following two production rules.

PERSON $\longrightarrow$ NAME AGE ADDRESS
PERSON $\longrightarrow$ NAME AGE ADDRESS PHONE-NUMBER

An instance of PERSON can correspond to either of the production rules and thus PHONE-NUMBER is an optional attribute for PERSON. In other words, people who have phones and those who do not can belong to the same class. This is not easily handled in many data models where either PHONE-NUMBER would be an attribute for every instance (thereby causing the problem with having to deal with null values), or the class PERSON would have to be modeled as a generalization of two different classes (which can be cumbersome). We can use a shorthand notation $A^{i,j}$ to denote at least i and at most j occurrences of the symbol A in a rule. With this notation the above two rules describing the PERSON class can be expressed by a single rule as follows.

PERSON $\longrightarrow$ NAME AGE ADDRESS PHONE-NUMBER^{0,1}

- **Lists vs. sets.**
  The grammar model is a list or sequence oriented model rather than a set-oriented model. One of the drawbacks of set-oriented models (like the relational and nested relational models) is that since objects in a set must have distinct values difficulties arise while expressing aggregation functions like average since duplicates are not allowed. Whereas a list-oriented model can handle both lists and sets.

- **A good basis for theoretical investigation.**
  It is fairly simple and has only a few basic constructs and would hence lend itself easily to theoretical investigation of modeling and query language issues.

- **Supports recursive data.**
  There are very few data models which allow for the specification of recursive data (a notable exception is the database logic introduced by Kuper and Vardi [17]). For example, consider a part–subpart database. A part can be represented by its name and other attributes (such as color, weight etc) and its subparts. What is important in this example it that a subpart is also a part, with a name, color, weight and subparts, etc. This situation can be conveniently modeled in the grammar model as follows:

Example 3
First, we define a part as

PART $\longrightarrow$ NAME COLOR WEIGHT SUBPARTS
SUBPARTS $\longrightarrow$ PART*

so notice that a part is recursively defined. A part-subpart database can then be modeled as

PART-SUBPARTDB $\longrightarrow$ PART*

3.2 Grammar Templates for Data Models

As seen above, most of the constructs supported by traditional data models can be supported in the grammar model. In this section we present examples of how databases represented in the nested relational model and the Format model have corresponding representations in the grammar model. We give restrictions, in the form of production templates, which guarantee that a grammar obeying them corresponds to a nested relational model or, respectively, to a Format model. Similar template restrictions characterize the ER, IFO and most other semantic models and the structural component of many object-oriented models.

In each of the following examples, the production rules that define the schema for the database are derived from a template which consists of a set of rules (which we will call *meta rules* to differentiate them from production rules of a grammar). This set of meta rules essentially describes the structure of a particular data model. Each meta rule in a template consists of terminal symbols (constants and symbols, i.e., character strings surrounded by angle brackets), variables (character strings not surrounded by angle brackets), and operators and parentheses (|, *, +, (, and)). In deriving the grammar rules for a schema from a template, each variable is instantiated by a value, (*e.g.*, an attribute name, a type name, etc). Terminal symbols are copied *verbatim*, without the surrounding angle brackets, into the derived grammar rule and are not interpreted. The operators |, +, and * in the meta rules are interpreted according to the usual convention in grammars, i.e., | represents alternatives, + represents at least one, * represents zero or more and parentheses are used for grouping.

3.2.1 Grammar Template for the Nested Relational Model

The following is a template of meta rules for the nested relational model.

NESTED-RELATIONAL-DATABASE <$\longrightarrow$> NESTED-RELATION^{+} (a)

NESTED-RELATION <$\longrightarrow$> TUPLE$^{<*>}$ (b)

TUPLE <$\longrightarrow$> (ATTRIBUTE | NESTED-RELATION)$^{+}$ (c)

ATTRIBUTE <$\longrightarrow$> ((<“>VALUE<”> <|>)* <“>VALUE<”>) | <STRING>|
<INTEGER> | <REAL> (d)

In this example the variables are NESTED-RELATIONAL-DATABASE, NESTED-RELATION, TUPLE, ATTRIBUTE and VALUE. The constants are <$\longrightarrow$>, < * >, <|>, <STRING>, <INTEGER>, <REAL>, <“>, and <”>. The operators are +, * and | and the parentheses ( ) for grouping. Meta rule (a) indicates that a nested relational database consists of a finite sequence of nested

relations. Meta rule (b) indicates that a nested relation consists of tuples. Meta rule (c) indicates that a tuple associated with a nested relation consists of a finite sequence of attribute values and nested relations. Finally, meta rule (d) indicates that an attribute has as a domain which is either a collection of constant values, the character strings, the integers, or the reals.

Example 4

The grammar derived from the above template for the students nested relation shown in Figure 3 is given below.

STUDENT-DB ⟶ STUDENTS
STUDENTS ⟶ STUDENT*
STUDENT ⟶ NAME COURSES-TAKEN
MAJOR STANDING
NAME ⟶ STRING
COURSES-TAKEN ⟶ COURSE-TAKEN*
COURSE-TAKEN ⟶ CNO SEMSTER GRADE
CNO ⟶ STRING
SEMESTER ⟶ "Fall 1984"| "Spring 1985"| ···
GRADE ⟶ "A" | "B" | "C" | "D" | "F"
MAJOR ⟶ "CSCI" | "MATH"
STANDING ⟶ "Fr" | "So" | "Jr" | "Sr" | "Gd"

The first rule in the above scheme is derived from the template by instantiating the variable NESTED-RELATIONAL-DATABASE with the value STUDENT-DB, copying the symbol ⟶ and instantiating NESTED-RELATION$^+$ with the relation name STUDENTS. The rest of the rules are derived similarly from the template.

STUDENTS

NAME	COURSES-TAKEN			MAJOR	STANDING
	CNO.	SEMESTER	GRADE		
BIII	C201	Fall 1984	A	CSCI	Jr
	C343	Spring 1985	B		
	C311	Fall 1985	B		
Mary	C201	Fall 1984	A	CSCI	So
	C311	Fall 1985	B		

Figure 3. An example of a nested relational database

The template for the relational model is very similar to the one for the nested relational model, the only difference being that attributes cannot have relations as values. So rule (c) in the template

for the relational model will be TUPLE <—→> ATTRIBUTE^{+}.

3.2.2 Grammar Template for the Format Model

The Format model was introduced by Hull and Yap [14] in an attempt to formalize semantic models. In particular, it combines in a concise manner the most basic components of data modeling. We selected the Format model because its relationship to the grammar model is particularly striking. Data types in the Format model are recursively defined in term of basic types (□–nodes) and the type constructors composition (or aggregation) denoted by⌓–nodes, classification (or generalization) denoted by△–nodes, and collection (or set formation) denoted by○–nodes.

Example 5

The schema of a Format database, shown in Figure 4, is taken from the paper by Hull and Yap [14].

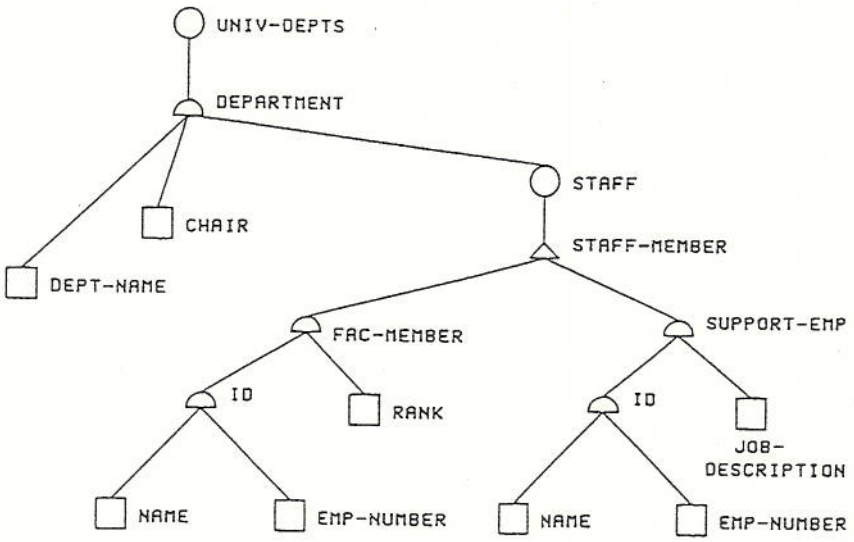

Figure 4. An example of the schema of a Format database.

The following is the template for the Format model (see also [10]).

FORMAT-DATABASE <—→> BASIC | COMP | CLASS | COLLECT
BASIC <—→> ((<“>VALUE<”> <|>)* <“>VALUE<”>) | <ALPHA> | <DIGIT*>
COMP <—→> (BASIC | COMP | COLLECT | CLASS)$^{+}$
CLASS <—→> (BASIC | COMP | CLASS | COLLECT) (<|> (BASIC | COMP | CLASS | COLLECT))$^{+}$
COLLECT <—→> (BASIC | COMP | CLASS | COLLECT)<*>

The first meta rule defines a Format database as a basic, composition, classification or collection type. The second meta rule describes the basic types. The third meta rule describes the composition constructor, the fourth describes the classification type constructor and the fifth describes the collection or set constructor.

The grammar derived for the Format database of Example 5 is:

UNIVERSITY-DB ⟶ UNIV-DEPTS
UNIV-DEPTS ⟶ DEPARTMENT*
DEPARTMENT ⟶ DEP-NAME CHAIR STAFF
DEPT-NAME ⟶ ALPHA
CHAIR ⟶ ALPHA
STAFF ⟶ STAFF-MEMBER*
STAFF-MEMBER ⟶ FAC-MEMBER | SUPPORT-EMP
FAC-MEMBER ⟶ ID RANK
ID ⟶ NAME EMP-NUMBER
NAME ⟶ ALPHA
EMP-NUMBER ⟶ DIGIT*
RANK ⟶ ALPHA
SUPPORT-EMP ⟶ ID JOB-DESCRIPTION
JOB-DESCRIPTION ⟶ ALPHA

Models like the ER model, the IFO model and most other known data models can also be modeled by the grammar model. Templates of meta rules that describe these models can be easily defined but are omitted here for brevity.

There are several benefits of using templates to derive schemes for data models.

- **Unify and Compare other models.**
  Although there are several semantic data models with powerful modeling constructs, there is no single data model that can satisfy the needs of all users. The grammar model on the other hand, can be customized easily to fit the specific requirements of an application by being able to generate the schemes of databases in various models from templates of grammar rules. It could thus serve as a unifying framework for supporting various data models. Since the grammar model can model various other data models, it can also be used as a basis for comparing the modeling powers of different data models.

- **Unify Heterogeneous Databases.**
  One of the main advantages of the grammar model is its ability to model data which is represented or stored in a variety of ways. It is therefore interesting to investigate how we can unify heterogeneous database applications into a single grammar database (see also [10]).

3.3 Modeling Hypertext in the Grammar Model

As shown in the previous section, the grammar model can be used to model various other database models like the relational model, the nested relational model and most semantic models. This means that all the database applications supported by these models can also be supported by the grammar model. For instance, the nested relational model is used to support applications involving complex objects like CAD/CAM, engineering designs, etc. Hence, these applications can also be supported by the grammar model. However, there are several applications for which there is no known model that can serve as an underlying basis. In this section we show how the grammar model can support one such application area, *viz*, hypertext systems, that is not easily supported by other data models.

Hypertext systems [6] provide greater flexibility and convenience than conventional text files by allowing complex organizations of information in the form of a network of nodes and links. A template for deriving the scheme of a hypertext system is given below.

DB <—> NODE<S> <LINKS>
NODE<S> <—> NODE$^{<*>}$
NODE <—> <NODE-NO> (ATTRIBUTE | NODE | NODE$^{<*>}$)* <FROM-LINK-NO*> <TO-LINK-NO*>
<LINKS> <—> <LINK*>
<LINK> <—> <LINK-NO> <FROM-NODE-NO> <TO-NODE-NO> ATTRIBUTE*
ATTRIBUTE <—> (($^{<«>}$VALUE$^{<»>}$ <|>)* $^{<«>}$VALUE$^{<»>}$) | <STRING> | <INTEGER> | <REAL>
<NODE-NO> <—> <INTEGER>
<LINK-NO> <—> <INTEGER>
<FROM-NODE-NO> <—> <INTEGER>
<TO-NODE-NO> <—> <INTEGER>
<FROM-LINK-NO> <—> <INTEGER>
<TO-LINK-NO> <—> <INTEGER>

A hypertext database (DB) consists of NODES and LINKS. A NODE has a NODE-NO which is essentially a node identifier. A NODE can have attributes that describe the node and it can also be an aggregation of other nodes. The links going to and emanating from a node are stored separately and references to these links (FROM-LINK-NO. and TO-LINK-NO.) are stored with each node. Each LINK in turn has a LINK-NO and references (FROM-NODE-NO. and TO-NODE-NO.) to the nodes that it connects.

Example 6

Figure 5 shows an example of a node-link representation of a book. Boxes represent nodes, arrows represent links and ellipses represent attributes. We do not show all the attributes in this figure, especially attributes like NODE-NO and LINK-NO which will be kept hidden from the user. The

grammar rules that define the scheme for this example, which are derived from the template given above, are shown below.

BOOK-DB ⟶ BOOKS LINKS

BOOKS ⟶ BOOK*

BOOK ⟶ NODE-NO NAME AUTHOR CHAPTER* FROM-LINK-NO* TO-LINK-NO*

NAME ⟶ STRING

AUTHOR ⟶ STRING

CHAPTER ⟶ NODE-NO CHAPTER-NO TEXT FROM-LINK-NO* TO-LINK-NO*

CHAPTER-NO ⟶ INTEGER

TEXT ⟶ STRING

LINKS ⟶ LINK*

LINK ⟶ LINK-NO FROM-NODE-NO TO-NODE-NO LINKTYPE

LINKTYPE ⟶ NextChapter | PreviousChapter | RefersTo

NODE-NO ⟶ INTEGER

LINK-NO ⟶ INTEGER

FROM-NODE-NO ⟶ INTEGER

TO-NODE-NO ⟶ INTEGER

FROM-LINK-NO ⟶ INTEGER

TO-LINK-NO ⟶ INTEGER

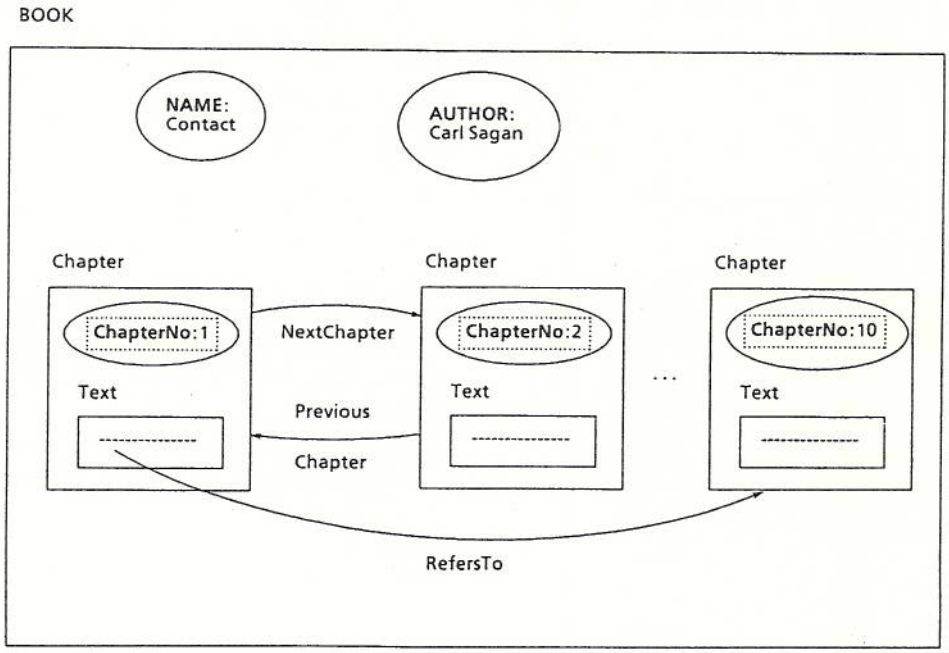

Figure 5. An example of a node-link representation

Let us suppose that the NODE-NO for Book is 1 and Chapters 1 through 10 have NODE-NO's from 2 to 11. Let the LINK-NO's of the links of LINKTYPE NextChapter and RefersTo emanating

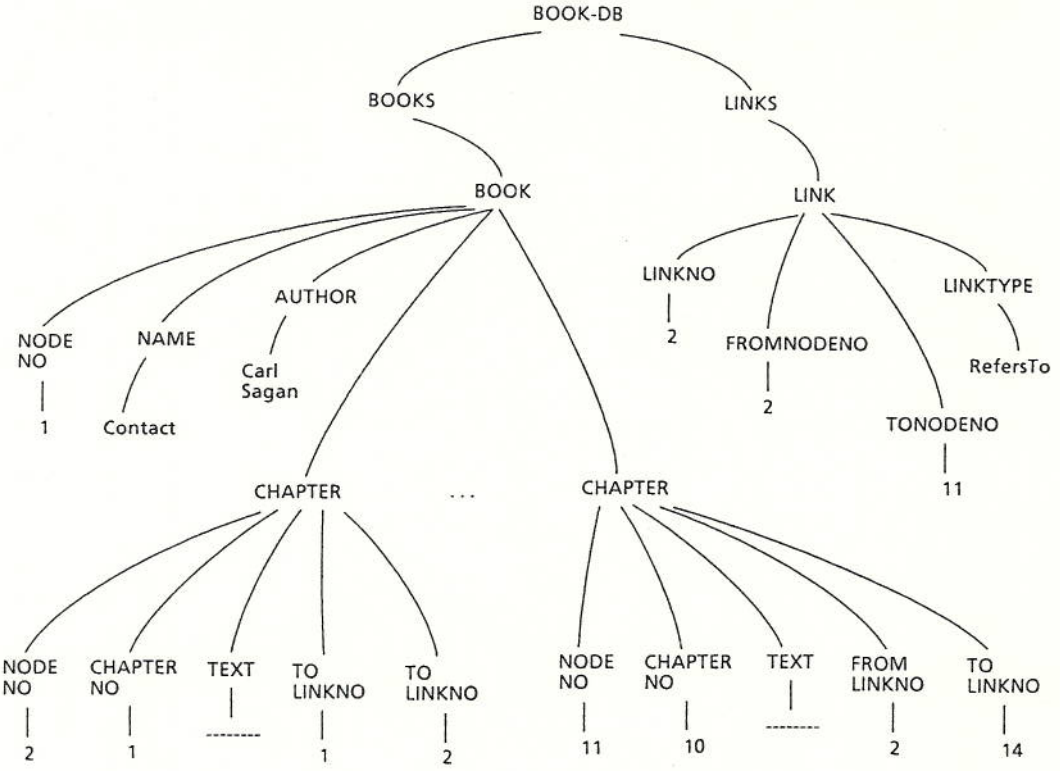

Figure 6. The p-string representation for the nodes and links in Example 6.

from Chapter 1 be 1 and 2 respectively. Figure 6 shows part of the p-string representation for this example of a node-link network represented by the rules shown above. While nodes and links can be modeled by many traditional database systems, there are some important and desirable features of hypertext that are not easily supported by traditional database systems. We list some of these features below and show how the grammar model could support all of these features. The first two features, namely, variable scheme and dynamic restructuring cannot be supported by most other traditional data models while the third feature stresses the need to have hypertext systems built on sound database models to facilitate searching and querying.

- **Variable Scheme.**
  Let us suppose that we want to create a database of books using a hypertext system. Now, books in general do not have a fixed format. A book may be divided into chapters, or it may be divided into several sections each of which is composed of chapters. A book may or may not have an index, a foreword, and so on. It is obvious then that models that can only support fixed schemes are not well suited for this example. The grammar rules given below show part of the top level node structure of a book. Links can exist between any two nodes but the link structure is not shown here. Although the rules shown here are by no means a complete description of possible schemes for books, they show how variable schemes can be easily represented in the grammar model.

BOOKS $\longrightarrow$ BOOK*

BOOK $\longrightarrow$ NODE-NO FROM-LINK-NO* TO-LINK-NO* NAME AUTHOR FOREWORD^1 †
(SECTION*|CHAPTER*) INDEX^1

SECTION $\longrightarrow$ NODE-NO FROM-LINK-NO* TO-LINK-NO* SECTION-NO CHAPTER*

CHAPTER $\longrightarrow$ NODE-NO FROM-LINK-NO* TO-LINK-NO* CHAPTER-NO TEXT

TEXT $\longrightarrow$ STRING

- **Dynamic Restructuring.**
  Users of many hypertext systems find that the static nature of the model makes it difficult or even impossible to reconfigure the structure of information if a need for such reconfiguration arises. Frank Halasz while describing some of the limitations of Xerox's NoteCards in [12] says "*In particular, a user in the very early stages of working with a particular set of information may not sufficiently understand the content and structure of that information. As the user's knowledge of the information space evolves, previous organizational commitments become obsolete.*" As an example consider the p-string shown in Figure 6 which represents the structure of a book that is composed of an Introduction, ten Chapters and an Index. Let us suppose now that the author of the book (who is still editing and rearranging parts of the book) decides to divide the book into two main sections each comprising five chapters. A set of select and transform operations can change the structure of the original p-string shown in Figure 6, yielding the p-string shown in Figure 7 and the resulting p-string would still conform to the grammar rules. While it may be argued that complex object models do support restructuring, like nesting and unnesting in the nested relational model, restructuring in these models produces new objects with a structure that has fewer or more levels of nesting than the original objects but does not really increase the semantic information in the database. In other words, restructuring is restricted to only changes in levels of nesting of the objects.

- **Search and Querying.**
  All hypertext systems support navigational access to information. This type of access alone is not sufficient as several nodes and links have to be traversed and searched before finding certain information and users often get lost while navigating through a complex network of nodes and links. Although browsers ease the task of navigation, they still do not speed up value based searches. Query based access is needed to complement navigation but most hypertext systems have inefficient search methods. A hypertext system that is based on a sound database model, like the grammar model, would lend itself more easily to searching and querying, since query languages [10, 11] can be defined that allow users to perform ad-hoc querying.

† Superscript i on a symbol indicates 0 or i occurrences of that symbol.

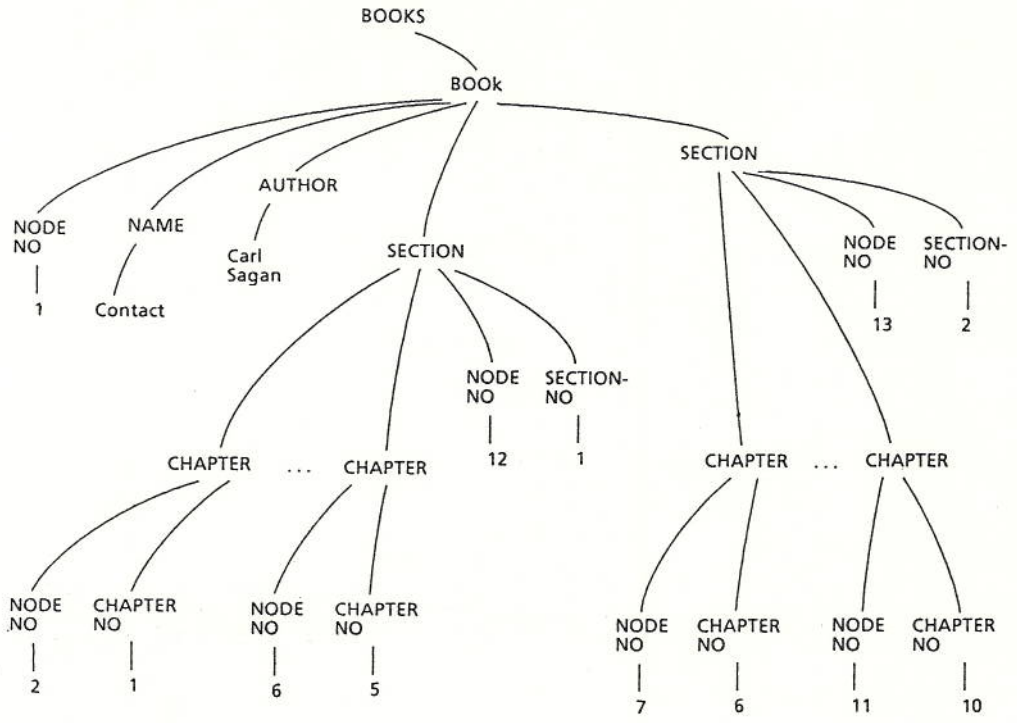

Figure 7. The result of restructuring the p-string in Figure 6.

4. Implementation Aspects

As indicated by Gonnet and Tompa [10], the implementation issues for grammar databases are a challenging task. We can think of a variety of ways to implement grammar models. As indicated in Section 3, certain grammar databases correspond directly with data specified in the Format model. So for such grammar databases, the implementation can be specified in terms of an implementation for Format databases. The problem of implementing Format databases can in turn be transformed into the problem of implementing nested or flat relational databases by mapping the format database to a nested or flat relational database. Thus the problem is reduced to that of implementing relational or nested relational databases. The problem with this approach is that it is not general since it is impossible to map every grammar database in a natural way to a Format database. A good example of this situation is the parts-subparts grammar database discussed in Example 3. In this section we therefore describe a possible way to implement general grammar databases. This approach is analogous to an implementation for nested relational databases [8]. In this approach, data is stored in two different data structures – (i) a VALTREE with value driven indexing capabilities for efficient processing of value oriented queries and (ii) a GENLIST (generalized list structure) for processing structure oriented queries. In addition, there is a main memory resident data structure, the CACHE, which performs the majority of the logical operations necessary to perform queries. Consider the following grammar and a p-string over this grammar.

Example 7

DB $\longrightarrow$ A*

A $\longrightarrow$ BC

B $\longrightarrow$ $f_1 \mid \ldots \mid f_k$

C $\longrightarrow$ D | E

D $\longrightarrow$ $d_1 \mid \ldots \mid d_l$

E $\longrightarrow$ (F | G)*

F $\longrightarrow$ $f_1 \mid \ldots \mid f_k$

G $\longrightarrow$ $g_1 \mid \ldots \mid g_m$

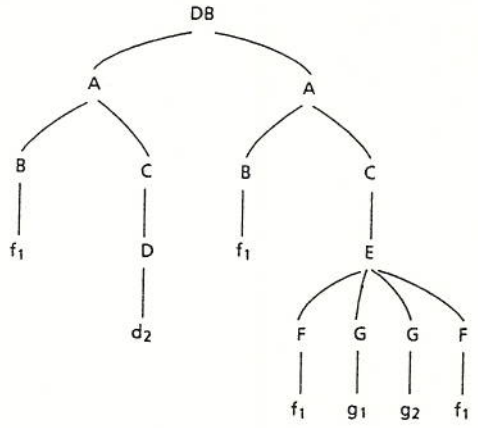

Figure 8. Example of a p-string of the grammar introduced in the text.

To denote objects in a p-string we need to define an addressing scheme for the nodes in that p-string. First we need the notion of a labeled p-string.

Definition Given a grammar G and a p-string g over G, we define the *labeled p-string* g, denoted *labeled*(g), as follows: If n is a $*$-node in g and c is a child of n, then label c as c_i in *labeled*(g), where i indicates that c is the ith leftmost child of n in the p-string g. If n is not a $*$-node, n retains its label (from g) in *labeled*(g).

Hence the labeled p-string corresponding to the p-string in Figure 8 is shown in Figure 9.

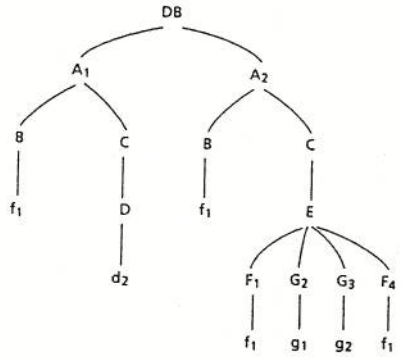

Figure 9. The labeled p-string corresponding to the p-string in Figure 8.

We next define the address of a node of a p-string.

Definition Let G be a grammar and let g be a p-string over G. Let n be a node in *labeled*(g), then the *address of n in g*, denoted *address*(n) is the path (denoted as a skewed list) from the root of *labeled*(g) to the node n.

address(DB) = (DB)

address(A_1) = (DB(A_1))

Let n be the node corresponding to the leftmost occurrence of the value f_1, then

address(n) = (DB(A_1(B(f_1))))

$address(d_2) = (DB(A_2(C(D(d_2)))))$
$address(g_2) = (DB(A_2(C(E(G_3(g_2))))))$

We are now ready to define the three data structures that form the basis of our implementation.

4.1 The VALTREE data structure

The VALTREE is a data structure which stores the values of leaf nodes in the grammar database along with the addresses. The VALTREE consists of three levels

1. The first level is the DOMAIN level. This level separates the non-compatible domains into separate subtrees.
2. The second level is the VALUE level. This level stores all the leaf node values according to the domain they belong to.
3. The third level is the ADDRESS level. This level stores the addresses of the values at the VALUE level.

The VALTREE of the labeled p-string in Figure 9 is shown in Figure 10.

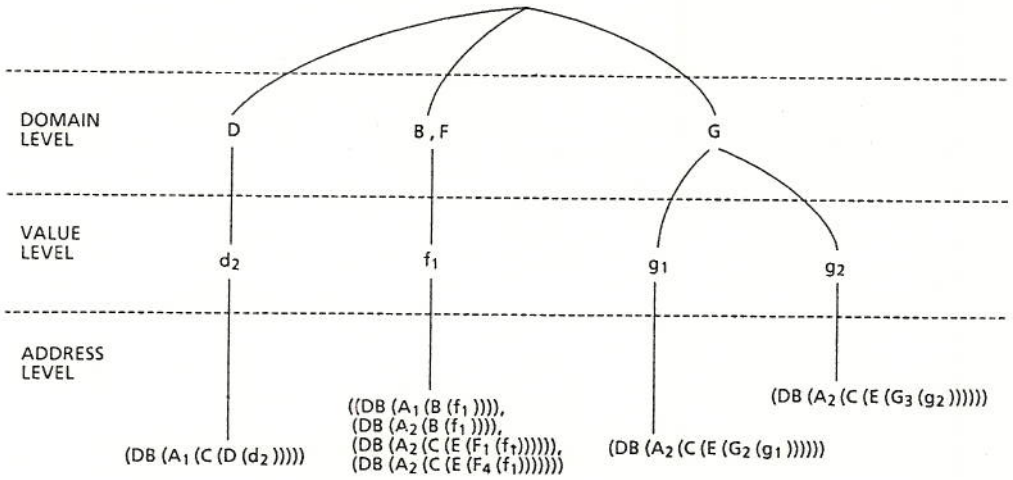

Figure 10. The VALTREE of the labeled p-string shown in Figure 9.

Notice that for the value f_1, we have four addresses. It is clear that there is overlap between these. This leads to a technique of merging addresses.

Consider the list of addresses corresponding to f_1, namely $(DB(A_1(B(f_1)))$, $(DB(A_2(B(f_1)))$, $(DB(A_2(C(E(F_1(f_1))))))$, $(DB(A_2(C(E(F_4(f_1))))))$. We can merge the information of these addresses and obtain the list $(DB(A_1(B(f_1))$, $A_2(B(f_1)$, $C(E(F_1(f_1),F_4(f_1))))))$.

This list can also be viewed as a tree. It corresponds to the maximal subtree of the p-string shown in Figure 9 with leaf nodes f_1. This observation allows us to view the ADDRESS level as having

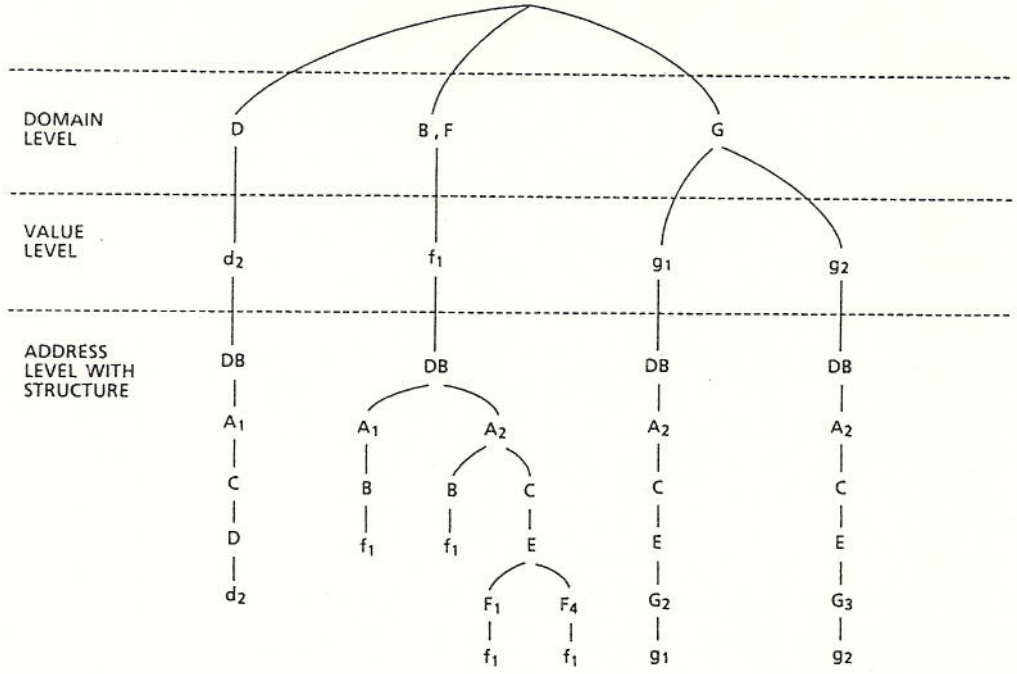

Figure 11. A VALTREE with structure at the ADDRESS level.

structure itself. Hence an alternative way to view the VALTREE of the p-string shown in Figure 10 is shown in Figure 11.

4.2 The GENLIST Data Structure

The GENLIST is a data structure which stores a p-string in a top-down fashion in such a way that given the address of a node (or a tree of addresses of nodes) the number of accesses to retrieve the objects corresponding to these addresses is minimized. Obviously, there are various ways to implement this data structure in secondary memory. We propose a data structure such that:

1. the data under a *-node of the p–string is organized in an index, so that efficient lookup is possible for any child of that *–node
2. data nodes in the p–string that are not *-nodes are clustered with their children, so that traversal from a node to its descendants can be done with the least amount of disk access.

4.3 The CACHE Data Structure

The CACHE is a (main memory) data structure which operates on (logical) addresses (and not directly on data†) as retrieved from the VALTREE. Its main function is to derive from these addresses and the formal grammar, the address description of objects that need to be retrieved from the GENLIST. It is clear that the CACHE data structure needs to be equipped with functions (or operators) which can take as inputs a) address trees from the VALTREE and b) information

† We have found that many queries can be handled in three phases: 1) index lookups (in the VALTREE), 2) index manipulation (in the CACHE) and 3) data materialization (using the GENLIST). Our implementation will be general however in the sense that if a query can not be broken down in these phases (i.e., we need to compute partial results), then partial data structures can be build.

from the grammar and derive object descriptions which can then be materialized by visiting the GENLIST with these descriptions. For more details about the advantages of the CACHE, we refer to [8].

To illustrate the three data structures, we give an example.

Example 8

Consider Example 7. Suppose we want to find the C objects associated with the B objects which have as a descendant the value f_1. We proceed by looking up in the VALTREE the address tree associated with f_1. This tree is shown in Figure 12.

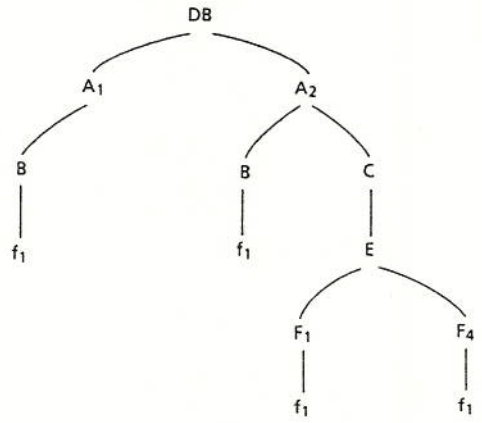

Figure 12. Address tree associated with f_1

In the CACHE we generate (by using appropriate operators) the maximal subtree with leafnodes corresponding to the object B. This results in the tree shown in Figure 13.

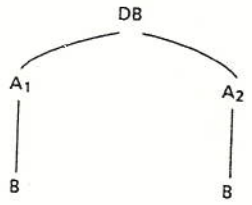

Figure 13. Maximal subtree with leafnodes corresponding to the object B

By looking at the formal grammar, we know then that we have to materialize the object (in list notation) $(DB(A_1(C(*)),A_2(C(*))))$ using the GENLIST. The resulting object is

$$(DB(A_1(C(D(d_2))),A_2(C(E(F_1(f_1),G_2(g_1),G_3(g_2),F_4(f_1))))))$$

Summary

We proposed to organize the implementations of grammar databases using three data structures, the VALTREE, the GENLIST and the CACHE. An important component of these data structure will be to develop operations that are relevant to them. The design of these operations will be guided by the various modes in which data is typically accessed, for example, providing support for a browsing strategy of access to data as well as support for ad hoc query processing. These operations will become the ingredients of an *access language* which will serve as the lowest level interface to the actual data in the grammar base. The next important research problem will then be to develop translation mechanisms to go from higher level data manipulation languages to this access language. An interesting problem will be the study of these translation algorithms and the possibility of optimization.

5. Future Research

This section outlines various areas of research in the context of the grammar model that need more investigation.

5.1 Development of Applications

Although there are a number of data models which can support several useful constructs for modeling databases, there can be no single model that can serve the needs of all database users. However, since we argued that the grammar model can support most of the common data modeling constructs, it can serve as the underlying basis for a generalized system that can support many different data models. In other words it could serve as the target model into which databases modeled in different source models are mapped. It would be interesting to see what models can be represented in the grammar model and also what models cannot be represented. We are currently investigating ways to map data and queries from different source models to the grammar model.

5.2 Data Definition

In all the research that we have done on the grammar model so far, we have considered a grammar database as a p-string over some grammar G without specifying any restrictions on G. So, as far as the grammar database is concerned, G can be an arbitrary grammar, *i.e.*, regular, context-free or context-sensitive. It appears, however, that most grammar databases can be restricted to be regular or context-free. We plan to investigate this claim in detail by determining to what extent database applications can be modeled with a regular or context-free grammar and see if there exist applications which need to be modeled as context-sensitive grammars.

5.3 Data Manipulation

An important part of any data model is its facility to manipulate data. Designing languages that

query and modify a database represented in the grammar model is an important area that needs to be investigated. There have been a couple of attempts to define such languages for the grammar model. Gonnet and Tompa [10] defined an algebraic query language with a functional flavor, especially targeted towards text databases. Gyssens, Paredaens and Van Gucht [11] defined an algebra and a rewrite language, (both with equivalent expressiveness), for a grammar model in which productions are in a normal form. These languages all have their advantages and disadvantages. The language of Gonnet and Tompa is defined in a rather ad hoc fashion and is particularly targeted towards textbases. On the other hand their language is very expressive and borders on being a full fledged programming language. The languages of Gyssens et. al., although applicable to all grammar databases, are not very powerful, *i.e.*, one can only emulate subsets of query languages that exist for set-oriented data models, such as, the relational and the nested relational algebra. We plan to generalize the language of Gyssens et. al., so that it becomes more expressive but retains it naturalness. We predict that this language will have a functional flavor since the objects that it manipulates are not sets but lists (see the section on implementation).

5.4 Constraints

Although we do not currently view the notion of constraints as one of our primary concerns, we plan to incorporate such notions in the grammar data model during our future research. The grammar model, as it is currently defined, allows many constraints, such as domain constraints and subset constraints to be incorporated into the production rules defining the scheme of the database. However, it is not clear whether constraints like functional dependencies can be built into the production rules without adding other constructs to the grammar model.

5.5 New Modeling Constructs

One of the limitations of the grammar model is that it is value oriented (like the relational model). A reference to an object can be made only by specifying its key values and cannot be made directly to the object itself. This limitation can be overcome by adding a new construct to the model that allows direct referencing. For instance in Example 6, references to nodes and links are made by using the attributes FROM-NODE-NO, TO-LINK-NO, etc., to specify the NODE-NO or the LINK-NO of the NODE or LINK that is being referred to. If we were to use a direct referencing mechanism, the production rules describing a LINK would be as shown below.

LINK $\longrightarrow$ FROM-NODE TO-NODE LINKTYPE
FROM-NODE $\longrightarrow$ @BOOK | @CHAPTER | @TEXT
TO-NODE $\longrightarrow$ @BOOK | @CHAPTER | @TEXT

where @BOOK denotes a reference to an object of type BOOK, etc. With this construct NODE-NO's and LINK-NO's are not required.

While the addition of more constructs would, undoubtedly, increase the functionality of the model, some of the advantages of the model might be lost as a result of increased complexity. For instance, fairly simple and straightforward query languages and storage structures can be defined on the existing grammar model; whereas, adding several new constructs could add further complexity to these issues. It would be interesting, however, to see just how much of an improvement one can make on this model while retaining all the advantages that the model has as it stands right now.

6. Summary

There are currently, a number of data models that have powerful modeling constructs designed to make the task of data modeling easier for database designers and users. However, there is no data model that can satisfy the needs of all users. We propose a generalized system with a model based on grammars as the underlying model that will allow data to be modeled in any one of several data models. Such a system would be able to meet the requirements of a several different types of database users and would support a very wide range of applications. In this paper we have pointed out several advantages of the grammar model over other known models. We have presented some initial ideas on transforming database schemes specified in one model to corresponding schemes in the grammar model. We have also described some of the implementation issues for the proposed system.

Acknowledgements

We thank Edward Robertson for his helpful suggestions and comments on an earlier version of this paper.

References

[1] ABITEBOUL, S., AND BEERI, C. On the power of languages for complex objects. Tech. rep., INRIA and Hebrew University, Sept 1988.

[2] ABITEBOUL, S., AND HULL, R. IFO: A formal semantic database model. *ACM Transactions on Database Systems 12*, 4 (December 1987), pp. 525–565.

[3] BANÇILHON, F., ET AL. The design and implementation of O_2, an object-oriented database system. In *Proceedings of the 2nd International Workshop on Object-Oriented Database Systems* (Bad Münster am Stein-Ebernburg Germany, 1988), pp. 1–22.

[4] BRODIE, M. L., AND RIDJANOVIC, D. Defining database dynamics with attribute grammars. *Information Processing Letters 14*, 3 (1982), pp. 132–138.

[5] CHEN, P. P. The entity-relationship model–toward a unified view of data. *ACM Transactions on Database Systems 1*, 1 (March 1976), pp. 9–36.

[6] CONKLIN, J. E. Hypertext: An introduction and survey. *IEEE Computer 9*, 20 (Sept 1987), pp. 17–41.

[7] COPELAND, G., AND MAIER, D. Making Smalltalk a database system. In *Proceedings of the Annual SIGMOD Conference* (Boston, Mass., June 1984).

[8] DESHPANDE, A., AND VAN GUCHT, D. An implementation for nested relational databases. In *Proceedings of the 14th VLDB Conference* (Los Angeles, California, 1988), pp. 76–87.

[9] FISHMAN, D. H., ET AL. Iris: An object oriented database management system. *ACM Transactions on Office Information Systems 5*, 1 (January 1987), pp. 48–69.

[10] GONNET, G. H., AND TOMPA, F. W. Mind your grammar-a new approach to modelling text. In *Proceedings of the 13th VLDB* (Brighton, England, 1987), pp. 339–346.

[11] GYSSENS, M. J., PAREDAENS, J., AND VAN GUCHT, D. A grammar–based approach towards unifying hierarchical data models. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (Portland, Oregan, 1989). (To be published).

[12] HALASZ, F. G. Reflections on Notecards: Seven issues for the next generation of hypermedia systems. *Communications ACM 31*, 7 (July 1988), pp. 836–852.

[13] HAMMER, M., AND MCLEOD, D. Database description with SDM: A semantic database model. *ACM Transactions on Database Systems 6*, 3 (September 1981), pp. 351–386.

[14] HULL, R., AND YAP, C. K. The Format model: A theory of database organization. *JACM 31*, 3 (1984), pp. 518–537.

[15] JAESCHKE, G., AND SCHEK, H. J. Remarks on the algebra on non-first normal form relations. In *Proceedings of the first ACM SIGACT-SIGMOD Symposium on Principles of Database Systems* (Los Angeles, 1982), pp. 124–138.

[16] KIM, W., BANERJEE, J., CHOU, H. T., GARZA, J. F., AND WOELK, D. Composite object support in an object-oriented database system. In *Proccedings of OOPSLA* (1987), pp. 118–125.

[17] KUPER, G. M., AND VARDI, M. Y. A new approach to database logic. In *Proceedings of the third ACM SIGACT-SIGMOD Symposium on Principles of Database Systems* (Waterloo, Ontario, 1984), pp. 124–138.

[18] SHIPMAN, D. W. The functional data model and the language DAPLEX. *ACM Transactions on Database Systems 6*, 1 (March 1981), pp. 140–173.

[19] THOMAS, S. J., AND FISCHER, P. C. *Nested Relational Structures.* JAIPress, 1986, pp. 269–307.