Kaleidoscope Data Model for An English-like Query Language

Sang K. Cha and Gio Wiederhold
Computer Science Department
Stanford, CA 94305, U.S.A.

Abstract

Most database interfaces provide poor guidance on ad hoc query formulation, burdening users to learn, and to recall precisely the query language and the database. Kaleidoscope avoids this problem by guiding the user's query construction actively. Based on a grammar specifying the syntax and semantics of an English-like Query Language (EnQL), the interface generates legitimate query constituents incrementally as menu choices. Additional intraquery guidance ensures the integrity of a partial query. The central theme of this paper is that to support Kaleidoscope's style of user-system interaction, the presence of a high-level data model is critical. The absence of an explicit model leads to ad hoc grammar design and query translation. Existing models are inadequate for supporting EnQL because of a significant conceptual gap between common English concepts and database representation of such concepts. This paper presents the features of Kaleidoscope, its data model for EnQL, and a mapping to the relational storage.

1 Introduction

The impedance mismatch between database languages such as SQL and host programming languages has motivated much research. Deductive and object-oriented database systems have emerged to provide a uniform language for application programmers [Minker, 1988, Zdonik and Maier, 1990]. While this research is expected to facilitate the development of database applications, yet another type of impedance mismatch exists between the end user's language and database languages: formal languages such as SQL force users to learn the syntax and semantics of the language and the underlying database, and to recall them precisely at the time of query formulation.

The impedance mismatch faced by end users cannot be treated by assimilating the database language to the user's language alone. The so-called natural language interfaces (NLIs) are intended to provide the user's habitual language. However, because of the difficulty of developing a large body of machine-interpretable knowledge on human linguistic behavior, these systems inevitably implement only seminatural languages and limited concepts. As a result, NLI users experience what is called proactive interference, the difficulty of remembering artificial constraints in a seminatural language [Shneiderman, 1980]. In a field evaluation of an NLI system with some 800 BNF rules, NLI users were reported to perform poorly relative to SQL users [Jarke *et al.*, 1985].

Kaleidoscope is a cooperative interface which relieves casual users of the impedance mismatch that they experience in interacting with database systems [Cha, 1991b]. It uses a grammar-driven menu system as a device for bridging the mismatch between the user's language and the database language. This system generates legitimate query constituents incrementally as menu choices, and users formulate a query by following menu guidance. A grammar specifying the syntax and semantics of a database language governs the system's automatic choice generation. While Kaleidoscope's grammar-driven menu guidance can improve the usability of formal query languages such as SQL [Cha, 1991a], a carefully designed English-like query language (EnQL) improves the efficiency of user-system communication significantly.

In supporting EnQL, Kaleidoscope takes a model-based approach. Instead of designing a grammar in an ad hoc manner, Kaleidoscope first defines a formal data model approximating the conceptual structure of restricted English queries. The grammar design then focuses on the unambiguous realization of references to individual model concepts, taking into account the capability of an underlying query processing system. One benefit of this model-based approach is that all queries created by the system's guidance are meaningful with respect to the underlying data model. In addition, the model-based approach provides a basis of defining a transportable interface architecture. This paper focuses on the data model for supporting EnQL.

In Section 2, we introduce the features of Kaleidoscope. Section 3 discusses the benefits of using EnQL, and its desired features. Section 4 elaborates Kaleidoscope's model-based approach. Section 5 describes Kaleidoscope's grammar and lexicon. Section 6 presents the data model for supporting EnQL. Section 7 presents an internal query language and a mapping to the relational storage. Section 8 briefly describes the implementation of Kaleidoscope. Section 9 compares our work with previous related work. Section 10 summarizes this paper.

2 Kaleidoscope User Interface

EnQL supports *wh*-queries beginning with *who*, *which*, *where*, *when*, and *how*. In Kaleidoscope, the user constructs an EnQL query incrementally from left to right. For example, the following query is created in ten steps of menu interaction:

[Q1] Who (1) wrote (2) ((8) which (3) 'DATABASE' (4) books (5) published by (6) 'McGraw–Hill' (7)) (8) since (9) 1982 (10)

The user may retract and change early selections. Figure 1 shows a few Kaleidoscope screen states encountered while creating Q1. Each state presents only choices that are both syntactically and semantically valid for extending a partial query. Database values, such as "DATABASE," "McGraw-Hill," and "1982," are created by selecting demon choices. Bounded by "<" and ">" on the screen, these choices prompt users with a hierarchical pop-up menu of database values or a type-in window constraining the user's input. Figure 2 exemplifies this by a pop-up menu of keywords. Choices marked with a triangle (▷) can be extended into submenus as shown in Figures 1 (c) and 2. Submenus organize related general and specialized terms hierarchically under a single choice. Some menu choices provide limited cues to the user in projecting the consequences of selecting them. For example, the prepositions "at" and "on" are ambiguous to the user. Kaleidoscope associates a documentation string with each choice to help users in projecting subsequent choice sets (Figure 1 (e)).

Control of Ambiguity The ambiguity of queries expressed in an English-like language is well-known. Domain-specific semantics and contextual information are helpful in reducing the number of possible interpretations, but do not guarantee the unique interpretation that both the user and the machine agree to.

Kaleidoscope takes the initiative in guiding users to avoid creating ambiguous queries. The menu window from which each token is selected provides the category information of the token. Overloading a choice with multiple interpretations is permitted only if the lexical ambiguity can be resolved by grammar. To avoid structural ambiguity, the system prompts users to enclose complex phrases with parentheses. For example, the query Q1, without the parentheses enclosing the object noun phrase, would be ambiguous because there are two possible interpretations on the scope of the phrase "since 1982." By providing the choice of parenthesizing complex phrases, as shown in Figure 1 (d), Kaleidoscope avoids the structural ambiguity.

Meaning-Based Guidance In parallel with choice generation, Kaleidoscope builds the meaning of a query incrementally for further intraquery conceptual guidance.

First, by executing the partial query, the system guides the user's value creation with a dynamically

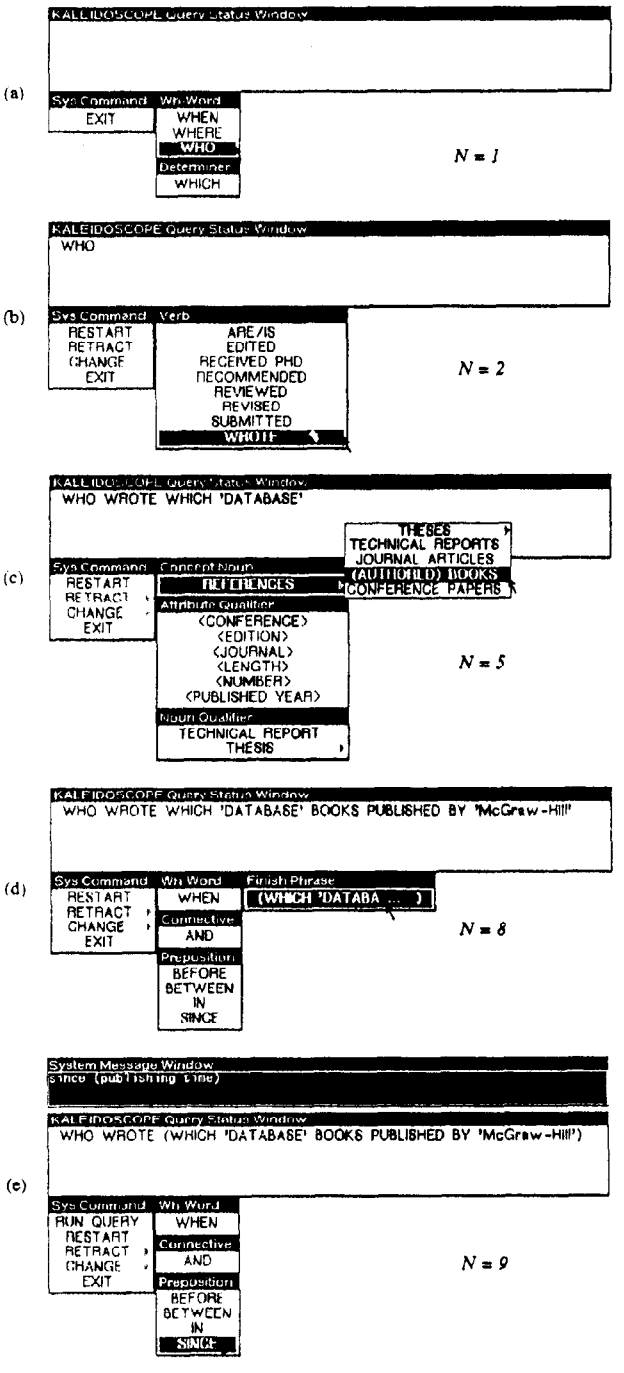


Figure 1: Progression of Kaleidoscope Screen States

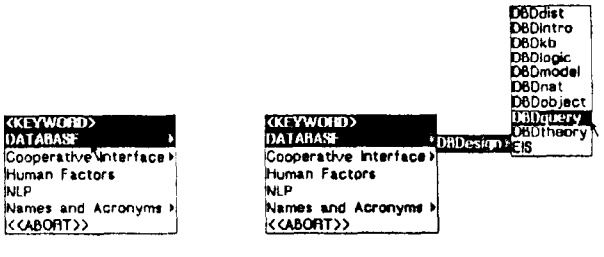


Figure 2: Two States of A Hierarchical Pop-Up Menu

computed range of values feasible for extending the partial query. This dynamic instantiation of pop-up menus not only narrows the range of choices for users but also reduces the chance of extensional query failure - the failure of syntactically well-formed queries to produce tuples due to the user's misunderstanding of database contents [Kaplan, 1982].

Second, by checking the meaning of a partial query against the system's knowledge of integrity constraints, the system detects user misconceptions in the middle of query composition. Consider a query

[Q2] Which (1) instructors (2) who (3) taught (4) CS445 (5)
since (6) 1980 (7) are (8) students (9) who (10) ...

to a university database with an integrity constraint:

[IC1] "Student instructors never teach CS 400 or higher level courses."

After the user's nineth selection, the system recognizes that the query Q2 becomes inconsistent with the integrity constraint, and warns the user of the inconsistency. This early detection of misconception, compared with postquery detection, saves the user's effort that would otherwise be wasted on completing a query bound to produce no meaningful result.

The same integrity constraint is also useful for generating informative messages. Consider IC1 rephrased as follows:

[IC2] "If an instructor teaches a CS course whose number is higher than or equal to 400, then the instructor is not a student."

Once the user finishes the fifth selection in the query Q2, the system derives a constraint that the instructor is not a student. This derived constraint is useful for guiding the user away from potential semantic inconsistency in the user's subsequent selections. However, in general, presenting all derived information may distract experienced users needlessly. The user has an option of disabling the system's presentation of derived informative messages.

Output Presentation Kaleidoscope presents each query result in a separate spreadsheet window to facilitate further screen manipulation. Figure 3 shows such a window containing the result of the query Q1. Relational, graph-drawing, and arithmetic operations are provided as generic spreadsheet functions. With this presentation strategy, EnQL does not need features for formatting and transforming query results.

The schema of a query result consists of the attributes drawn from the entity sets specified by *wh*-words. To determine these projection attributes, the system either prompts the user with a pop-up menu of selected entity attributes or takes default projection attributes defined by the schema.

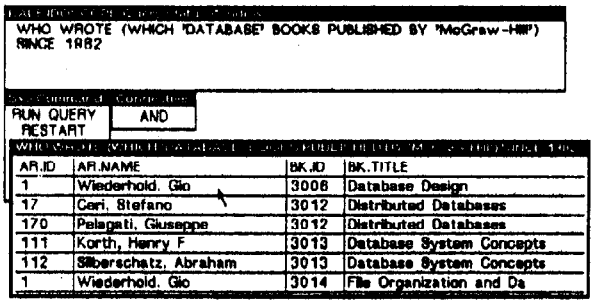


Figure 3: Kaleidoscope Screen with Query Output

3 Why An English-like Query Language?

Although grammar-driven menu guidance can improve the usability of formal languages such as SQL [Cha, 1991a], EnQL enables the user to express queries more concisely than SQL, and relieves the user of transforming mental queries to those based on a underlying database implementation. Some examples show that SQL translations of EnQL queries may consume ten times as many tokens, and involve joins of several tables. For example, while the query Q1 is created in ten steps, its SQL translation with no redundancy, comprises 89 tokens, not counting SQL punctuation marks ("." and ","). Figure 4 shows this SQL query. One EnQL token 'DATABASE' alone accounts for 12 SQL tokens.

The number of tokens required to express a query is critical to the performance of query interface users. In the absence of other information on the complexity of a query, the ratio of the required number of tokens and the capacity of human short-term memory (7 ± 2 [Miller, 1956]) could be a measure of the user's cognitive burden. The higher this ratio, the more cognitive swapping, we suspect, is required to produce a query. The fact that EnQL queries consume a significantly smaller number of tokens than its SQL translations suggests that EnQL queries are more efficient and probably easier to phrase than SQL queries. This leads us to ask what the elements of the English-likeness are that contribute to the conciseness of EnQL queries, and that will be of further benefit to grammar-driven menu interface users. Our answer to this question is summarized in terms of four degrees of freedom:

1. *Distribution of modifiers* over the span of a sentence: A underlying data model should support a rich set of modifier types. This set includes verb modifiers as well as prenoun and postnoun modifiers.
2. *Reduction of query production steps*: For example, the user should be able to choose a shorthand expression "DATABASE books" instead of its full-fledged version "books written on DATABASE." The latter structure is still needed for the user to add adverb phrases modifying the verb. Similarly,

```
SELECT   ar1.author, p2.pname,
         b9.id, reference7.title,
FROM     author_reference ar1,
         person p2
         keyword_view kv4,
         reference_keyword rk5,
         reference r7,
         book b9,
         organization o10,
         publish_reference pr11,
WHERE    pr11.reference = r7.id and
         o10.id = pr11.organization and
         b9.id = pr11.reference and
         b9.id = r7.id and
         b9.id = rk5.reference and
         kv4.id = rk5.keyword and
         ar1.reference = b9.id and
         ar1.author = p2.id and
         o10.name = 'McGraw-Hill' and
         r7.year >= 1982 and
         kv4.string = 'DATABASE'
```

Figure 4: SQL Translation of Q1

the phrase "McGraw-Hill's books" is a shorthand for "books published by McGraw-Hill."

3. *Alternative ordering of references*: While users can tolerate limited syntax with menu guidance, too restrictive a syntax forces users to navigate through a narrow network of choice sets. To lessen this burden, the language syntax should support means of alternatively ordering references to entities and relationships, such as passive voice.

4. *Choice of commitment degree*: The user's selection of a generalized term, compared with a specialized term, makes a weak commitment in referring to entity sets and relationships, thus leaving more options in the subsequent menus. For example, the choice set following "which persons" includes verbs that are not applicable to "which authors," such as editing books.

4 Model-Based Approach

To support Kaleidoscope's style of user-system interaction, coupling of syntactic and semantic information is indispensable. The lack of semantics in choice generation results in the failure to prune irrelevant choices, which not only misleads users toward nonsensical queries but also wastes the screen space and potentially increases the user's choice search time. On the other hand, in defining Kaleidoscope's architecture, we are concerned with the ease of creating specific database interfaces. It is desirable for the architecture to possess the following features: (1) a domain-independent grammar, (2) a domain-independent translator, and (3) ease in generating a domain-specific lexicon, where the lexicon refers to a collection of categorized choices.

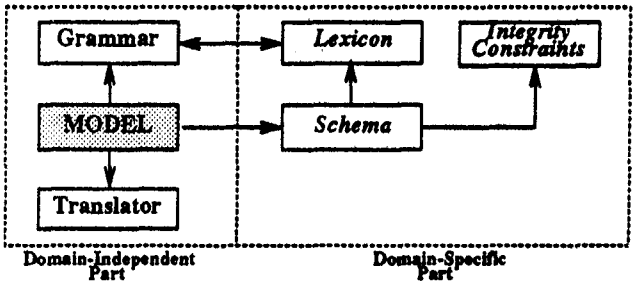


Figure 5: Model-Based Approach

The central theme of our approach is that in seeking these architectural goals, the presence of a high-level data model formalizing the conceptual structure of EnQL queries is critical. The absence of an explicit model leads to ad hoc grammar design and query translation, thus harming the transportability of the system. Existing data models are not adequate for supporting the desired features of EnQL. There is a significant conceptual gap between common English concepts and database representation of such concepts. Ignoring this gap would either force users to create cumbersome queries, or overload the grammar with a complex mapping to achieve a comfortable level of English-likeness.

Figure 5 shows how the model serves as the basis of defining the run-time components of a transportable grammar-driven menu system. In this model-based approach, grammar design focuses on the specification of rules for unambiguously realizing references to model concepts, and dynamically constructing query meaning. Unlike the conventional NLI approach in which the goal of non-normative systems is pervasive [Bobrow and Bates, 1982, Marcus, 1982], the normative design principle [Simon, 1981] is applied. The design process sets a target expressive power by considering the capability of the underlying query processing system. Alternative designs are evaluated by a cost function. In our research, we have taken conjunctive queries as the target expressive power, and devised a simple cost model of user query production when using grammar-driven menu interfaces [Cha, 1991b]. One benefit of this model-based grammar design is that all queries created via menu guidance are meaningful with respect to the data model.

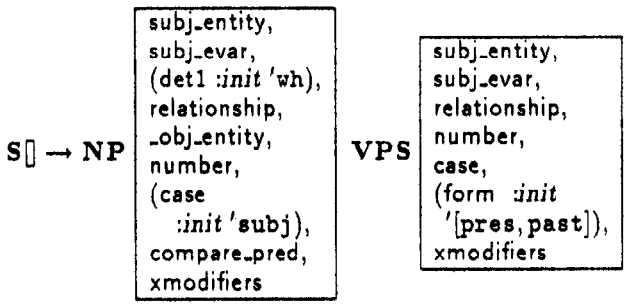
Based on grammar specification, we design a set of procedures that generate the lexicon automatically from the schema. This relieves interface creators of dealing with the linguistic part of the system. Finally, with a well-defined model, it is possible to define a mapping from this model to a target storage model. We assume the relational model for the storage model.

5 Grammar and Lexicon

This section briefly describes Kaleidoscope's grammar and lexicon. A more detailed description will be pre-

sented in [Cha, 1991b].

Around 40 phrase structure rules specify EnQL. The following shows a top-level rule prescribing the construction of a class of queries comprising a noun phrse (NP) followed by a conjunction of verb phrases (VPS):



Each grammar symbol is augmented by a collection of feature attributes (shown in boxes) that formalizes the context of constituent structures. Both syntactic and semantic features are captured this way. The runtime bindings of these features come primarily from the lexicon, although the grammar rules often provide domain-independent values such as wh and subj. Feature attributes may take a limited constraint formula: disjunction of atoms (enclosed by "[" and "]"), negated disjunction of atoms, or conjunction of one disjunction and one negated disjunction. Unification of feature bindings is enforced between a parent rule and its children to block unnecessary application of child rules. We call the feature attributes of nonterminals *context variables* because they can be manipulated by a few types of procedural decorations. These decorations, activated by the events of Kaleidoscope's chart-based grammar interpreter, initialize unbound context variables, construct the partial query meaning, and interface the system's nonlinguistic part.

The lexicon consists of a list of preterminal categories. Each category defines a list of feature attributes, a list of choices, and a display menu window. Table 1 shows sample lexicon entries. Semantic feature attributes, such as entity, v_subj, and v_obj, refer to the schema concepts.

WH-PN:	
"who"	
entity =	*Person*
v_subj =	[*Author_Reference*, *Edit_Book*, ...]
v_obj =	NIL
V:	
"wrote"	
rel =	*Author_Reference*
subj_entity =	*Author*
obj-entity =	*Reference* ∧ $\overline{\textit{Edited\_Book}}$
arity =	2
tense =	past
form =	past
ENTITY-SET-N:	
"books"	
entity =	*Book*
v_subj =	NIL
v_obj =	[*Author_Book*, *Edit_Book*, *Publish_Book*]
countp =	plus
number =	pl
"(authored) books"	
entity =	*Authored_Book*
v_subj =	NIL
v_obj =	*Author_Book*
countp =	plus
number =	pl

Table 1: Sample Lexicon Entries

6 The Data Model

6.1 Basic Concepts

Entities, *relationships*, and *relationship modifiers* describe the overall schema of a database. Entities correspond to noun phrases (NPs) appearing as subjects, objects, and prepositional phrase (PP) objects. Relationships model domain-specific verbs, and take one or two entities as arguments. Relationship modifiers represent adverb phrases, such as *wh*-adverbs and prepositional phrases. Each relationship modifier takes two arguments: one for the base entity involved in modifying the relationship, and another for the relationship that it modifies. The arguments of both relationships and relationship modifiers can be specified by constraint formulas as well as atoms. In our model, a typical E-R relationship [Chen, 1976, Chen, 1980] is represented by a relationship of fixed arity (≤ 2) and an arbitrary number of relationship modifiers.

Figure 6 shows a graphically represented schema. Rectangles, diamonds, and trapezoids represent entities, relationships, and relationship modifiers, respectively.

6.1.1 Entities

Entities model not only objects with unique identity such as *Author* and *Book* but also mass nouns, such as *Salary*, if domain-specific verbs take them as subjects, objects, or PP objects. Count and mass entities have different *wh*-determiners in EnQL: "which" and "how much," respectively. Mass entities may have comparative adjectives as in "Who earn more salary than their managers?" An entity definition includes:

- a feature *countp*, which indicates the countability of an entity,
- a noun to be used for reference,
- a set $\mathcal{A}$ of attributes (or properties),
- a set $\mathcal{K}$ of key attributes ($\mathcal{K} \subseteq \mathcal{A}$),
- a set $\mathcal{P}$ of default projection attributes ($\mathcal{P} \subseteq \mathcal{A}$).

Entity Attributes An attribute is marked to indicate if it is qualified for a prenoun modifier. Key attributes are in general not allowed to appear as prenoun modifiers. All attributes may appear in postnoun modifier clauses. Each entity attribute refers to

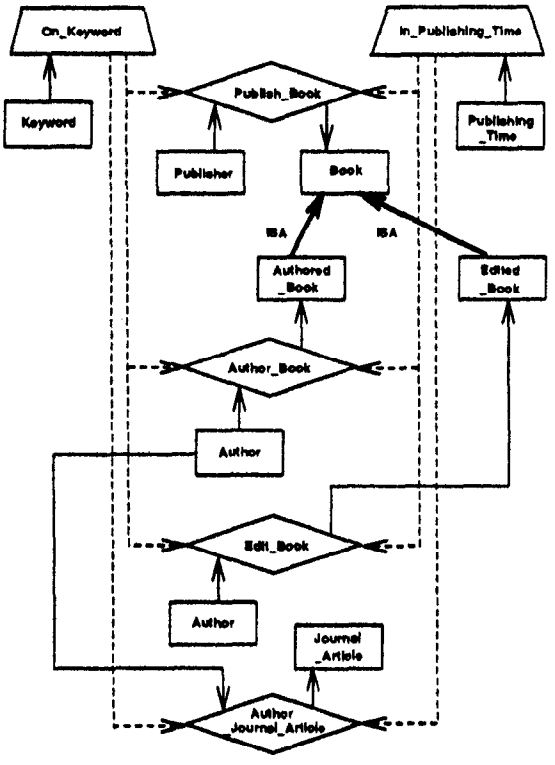


Figure 6: A Graphical EnQL Schema

a *domain* and has a noun for its reference. A domain definition contains information on guiding the user's value creation, such as the type of pop-up menus.

6.1.2 Relationships

Relationship arguments are assigned their roles: subject or object. In the graphical schema representation, an arrow from an entity to a relationship indicates that the entity plays the subject role, while an arrow from a relationship to an entity indicates that the entity plays the object role. A relationship definition additionally includes:

- a feature *tense* to specify the legitimate tenses of a relationship,
- a verb to be used for reference.

Example The query Q1 illustrates the realization of two binary relationships: *Author_Book* ($Author_{subj}$, $Authored\_Book_{obj}$) and *Publish_Book* ($Publisher_{subj}$, $Book_{obj}$) underly the verbs "wrote" and "published," respectively.

The unary relationship *Receive_PhD* ($Author_{subj}$) with *tense* = *past* models the fact that some authors received PhD. When this relationship is realized in a query, an NP referring to *Author* appears in the subject position of verb "received PhD."

While three-place relationships are conceivable to model ditransitive verbs (e.g., "x pays y \$$z$"), such relationships are substituted by two-place relationships by moving indirect objects to adverb positions ("x pays \$$z$ to y).

6.1.3 Relationship Modifiers

An arbitrary number of relationship modifiers may be associated with each relationship, and vice versa. As a result, the relationship argument of a relationship modifier is typically specified by a disjunction of relationships. For example, in Figure 6, *In_Publishing_Time* and *On_Keyword* modify the relationships *Author_Book*, *Edit_Book*, *Publish_Book*, and *Author_Journal_Article*. The relationship argument of these modifiers is then expressed by a disjunctive formula:

$$Author\_Book \vee Edit\_Book \vee Publish\_Book \vee Author\_Journal\_Article.$$

A relationship modifier may be realized with multiple prepositions. For example, although our convention affixes a representative preposition "In" to the base entity name "Publishing_Time," *In_Publishing_Time* may be realized not only as "in 1982" but also as "since 1982" or "before 1982."

Some relationship modifiers are shared by a set of relationships in the sense that two verbs sharing an NP also share adverb phrases. The query Q1 exemplifies this. If either of two verbs "wrote" and "published" is restricted by the adverb phrase "since 1982," the other is also restricted by the same adverb phrase. This information is useful for checking the semantic consistency of two related verb phrases. In the graphical schema representation, arrowed lines connect relationship modifiers to relationships. If the line ends with multiple relationships, they share the relationship modifier.

6.2 ISA Hierarchies

ISA hierarchies organize schema concepts by similarity and difference. Figure 6 also shows *ISA* relationships between entities. The semantics of the entity hierarchy is that if E_d is a descendent of E_a ($ISA(E_d, E_a)$), then E_d is a subset of E_a. E_d inherits all attributes of E_a, and may define new attributes. The model imposes a mandatory rule regarding entity specialization:

> *If a set of relationships disjointly divides an entity set, create specialized entity sets, one for each of the relationships.*

For example, two relationships *Author_Book* and *Edit_Book* disjointly divide the entity set *Book* because a book is either authored or edited but not both. (An edited book, however, may contain many authored chapters or articles.) By the mandatory rule, two entities *Authored_Book* and *Edited_Book* are created as specializations of *Book*, and used for specifying the object arguments of the relationships. This mandatory entity specialization avoids nonsensical queries such as "Who wrote books edited by ..." Here, the entity *Authored_Book* referred to by "books" cannot be an argument of *Edit_Book*. Thus "edited by" is pruned

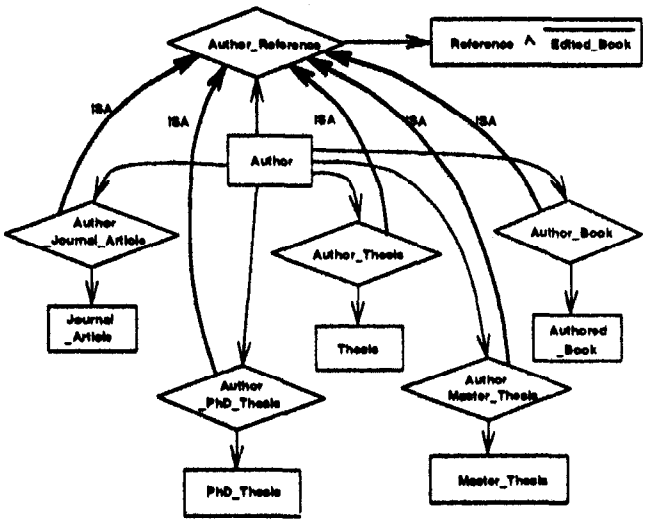


Figure 7: A Relationship *ISA* Hierarchy

from the choice set presented after the user's selection of "books."

The entity hierarchy enables users to query a specialized entity set. Let N_a and N_d be NPs realizing entities E_a and E_d. Users may ask:

"Which N_a are N_d?"

("Which theses are PhD theses?")

Note that reversing the order of N_a and N_d leads to trivial questions such as "Which PhD theses are theses?" Therefore EnQL does not support this type of query.

Relationships are also organized into hierarchies as shown in Figure 7. The arguments of a parent relationship subsume the arguments of all of its child relationships. A child relationship inherits all the modifiers associated with its parent. Additional relationship modifiers may be defined for the child relationship. The existence of additional relationship modifiers mandates relationship specialization. For instance, the relationship *Author_Journal_Article* is specialized from *Author_Reference* because the modifier *In_Journal* is applicable only to *Author_Journal_Article*. A relationship is also specialized without introducing new modifiers when its arguments are specialized. For example, *Author_Thesis* has specializations *Author_PhDThesis* and *Author_MasterThesis*.

Benefits The *ISA* hierarchies form a basis for:

- Extending unification in such a way that two atoms in *ISA* relationship unify to the specialized one. As a result, *Book* and *Authored_Book* unify to *Authored_Book*.
- Overloading attribute-based choices. For example, the choice "edition" is specified as the attribute of the entity *Book*, but also serves as an attribute of *Authored_Book* and *Edited_Book*, thus reducing the number of choices on the screen.
- Supporting the user's choice of commitment degree: Let $\mathcal{V}_x$ denote a set of verbs that can be attached to an NP referring to the entity E_x. Then $\mathcal{V}_d \subseteq \mathcal{V}_a$ holds for $ISA(E_d, E_a)$. Similarly, let $\mathcal{A}_x$ be a set of attributes that can be matched by the entity specification E_x. Then, $\mathcal{A}_d \subseteq \mathcal{A}_a$ holds as well.
- Presenting general/specialized terms hierarchically on the menu.
- Organizing lexicon entries hierarchically such that the failure of unification at a nonleaf node guarantees the failure at all of its descendents. For example, if the match fails at *Books*, it is unnecessary to try to match *Authored_Book* and *Edited_Book*.

6.3 I-OVERLAP

Often two entity sets such as *Thesis* and *Technical_Report* overlap, even if they are not in *ISA* relationship. The *I-OVERLAP* relationship captures such intrinsically overlapping entity sets. This relationship provides the basis of determining legitimate noun qualifiers, such as "thesis" in "thesis technical reports," and qualified NP complements for establishing the entity identity as in "Which technical reports are 'Stanford' theses?" *I-OVERLAP* has following properties:

Symmetry:
$I\text{-}OVERLAP(E_1, E_2) \Rightarrow I\text{-}OVERLAP(E_2, E_1)$. Thus, if "thesis technical reports" is legitimate, so is "technical report theses."

Pseudotransitivity:
$ISA(E_1, E_2) \wedge I\text{-}OVERLAP(E_2, E_3) \Rightarrow I\text{-}OVERLAP(E_1, E_3)$. As a result, "PhD thesis technical reports" is also a legitimate NP.

With the *I-OVERLAP* relationship, entity sets with multiple parents are not necessary. As a result, the *ISA* hierarchies in our model retain the simplicity of tree structures. Compared with the lattice-based multiple inheritance, our inheritance model reduces the number of entity sets to represent in the schema significantly.

6.4 Derived Attributes and Subordinate Entities

It is desirable for entities, relationships, and relationship modifiers to be defined without redundancy. For example, if *Keyword* is modeled as the base entity of the relationship modifier *On_Keyword*, the keyword information does not appear in the definition of *Book*. However, to support the shorthand expression "DATABASE books," it is desirable to treat *Keyword* as if it were an attribute of *Book*. In Kaleidoscope, this is done by *derived attributes*: the attributes from a relationship modifier's base entity are imported to the argument entity of the relationship.

Similarly, it is desirable to refer to some entities as if they were subordinate to others, as in "which publisher's books" or "which book's publishers." The relationship *Publish_Book* is implicit in both cases. The

schema may define an argument entity of a binary relationship subordinate to the other.

7 Internal Query Language

This section defines an internal query language (IQL) for representing the query meaning and integrity constraints, and presents a mapping from our model to the relational model.

Let R, M, and S be the sets of symbols representing relationships, relationship modifiers, and built-in predicates. S includes $=$, $\neq$, $>$, $<$, $\geq$, $\leq$, *between*, and *not between* as its members. Note that S is closed under negation. A query may contain range-restricted variables for entity sets and relationships.

7.1 Query Meaning Representation

A query Q is a conjunction of positive literals P_i:

$$Q = \{(e_1, e_2, \ldots) \mid \bigwedge_{i=1}^{n} P_i\}$$

where $e_1, e_2, \ldots$ are free entity variables. Let p_i be the predicate symbol of P_i, then p_i is drawn from R, M, or S. All relationship variables, and the entity variables which do not appear as free variables are existentially quantified. The following condition holds for Q: for each literal P_i, there exists at least one literal $P_j (i \neq j)$ such that the variables in the arguments of P_i and P_j overlap.

EnQL grammar encodes the following mapping from EnQL to IQL:

- For each reference to an entity set or an individual entity in EnQL, an entity variable is created. If an entity set reference is qualified by a *wh*-word, the variable is free; otherwise, it is existentially quantified.
- If $p_i \in R$, P_i has three arguments: a relationship variable, and variables for the subject and object entities. If p_i represents a unary relationship, one argument is left empty.
- If $p_i \in M$, the first argument of P_i is a relationship variable, and the second is a variable for the base entity of p_i.
- If $p_i \in S$, the first argument of P_i is a pair of an entity variable and an attribute. Either constants or pairs of entity variables and attributes are qualified for the remaining arguments.

Example The system builds the following conjunctive query incrementally while the user creates the query Q1.

$$\{(x, y) \mid (Author\_Book\ r_1\ x\ y) \wedge (=\ y.keyword\ \text{"DATABASE"}) \wedge (Publish\_Book\ r_2\ p\ y) \wedge (=\ p.name\ \text{"McGraw-Hill"}) \wedge (In\_Publishing\_Time\ r_1\ t) \wedge (\geq\ t.year\ 1982)\}$$

where $x \in Author$, $y \in Authored\_Book$, $p \in Publisher$, $t \in Publishing\_Time$, $r_1 \in Author\_Book$, $r_2 \in Publish\_Book$.

In this query, variables p, t, r_1, r_2 are existentially quantified.

7.2 Integrity Constraints for Intraquery Cooperation

An integrity constraint is a negated conjunction of literals:

$$\neg (\bigwedge_{i=1}^{m} L_i)$$

where the predicate symbol l_i of L_i is drawn from R, M, or S. We restrict L_i to be positive if l_i belongs to R and M. With S closed under negation, the literals based on the symbols in S could be treated as either positive or negative.

To illustrate, the integrity constraint IC1 is formally expressed as:

$$\neg ((Teach\ r\ i\ c) \wedge (I\text{-}OVERLAP\ i\ s) \wedge s \in Student \wedge (=\ c.dept\ \text{"CS"}) \wedge (\geq\ c.number\ 400))$$

Here, for the simplicity of presentation, we left out the declaration of range variables except for s. If all literals of an integrity constraint are true in a query, the system warns the user of the integrity constraint violation. For this inference, Kaleidoscope uses OPS5 [Forgy, 1981]. Thus, Kaleidoscope represents the above integrity constraint by a production rule:

IF $(Teach\ r\ i\ c) \wedge$
$(I\text{-}OVERLAP\ i\ s) \wedge s \in Student \wedge$
$(=\ c.dept\ |CS|) \wedge (\geq\ c.number\ 400)$
THEN MAKE (*Warning* |Student instructors never teach CS 400 or higher level courses|).

The Integrity constraint IC2, on the other hand, has a different type of THEN part to derive literals:

IF $(Teach\ r\ i\ c) \wedge$
$(I\text{-}OVERLAP\ i\ s) \wedge$
$(=\ c.dept\ |CS|) \wedge (\geq\ c.number\ 400)$
THEN MAKE $s \notin Student$

7.3 Mapping to Relational Storage

7.3.1 Mapping

A mapping from an EnQL schema to a relational schema consists of:

- An attribute-level mapping, which maps entity attributes to database (DB) relation attributes.
- An entity-level mapping, which adds restrictions and joins to the collection of attribute-level mappings.
- A relationship-level mapping, which defines joins between the arguments of a relationship, and between the argument of a relationship and the base entities of its modifiers.

7.3.2 Query Translation

Query translation proceeds as follows:

1. From an IQL representation, create an input record with:
   - Entity variables and their types.
   - Free entity variables and projection attributes: the latter are acquired either by prompting the user with pop-up menus or by retrieving the default set of attributes.
   - Entity restrictions and joins.
   - Relationships and their modifiers: collect relationship modifiers within the relationship that they are associated with. Relationship variables are removed in this process.
2. Create a hash table for keeping track of entity views. An entity view is created for each entity variable in the query and contains a minimal list of attributes: key attributes, projection and restriction attributes, and join attributes that are either explicit in the user query or required by the entity and relationship-level mapping.
3. For each relationship R with a list of modifiers $\{M_i\}$, do the following:
   (a) Collect entity views corresponding to the entity variables found in R and M_i's.
   (b) Create DB joins across these entity views as defined by the relationship-level mapping. Also unify multiple instantiations of identical DB relations in this process.
4. For each instance of *I-OVERLAP* in the query, create an equijoin of two entity views.
5. Collect the DB projection attributes, DB relation instances (pairs of relation names and unique identifiers), DB table joins, and DB table restrictions, and create an SQL query.

8 Implementation of Kaleidoscope

Kaleidoscope runs on a XEROX 1186 LISP machine with a SUN configured as a remote database server [Cha, 1991a]. Its interpreter is made up of two interacting procedures:

- A chart-based choice generator interprets grammar and incrementally generates choices. A runtime structure called *chart* keeps track of alternative parses and pending hypotheses about a partial query [Kay, 1980] and context variable bindings. The unification embedded in this interpreter recognizes the generalization/specialization hierarchy.
- An intraquery conceptual guidance module based on OPS5 keeps track of the partial query meaning and generates corrective and informative messages.

Kaleidoscope manages the schema, integrity constraints, and the dictionary of words for referring to schema concepts in the relational database. This DBMS-based approach provides the locality of changes and the set-oriented querying capability over the knowledge structure. Furthermore, with rules and triggers supported as part of DBMS functionality [Sybase, 1988, Stonebraker *et al.*, 1987, Widom and Finkelstein, 1989], updates can be automatically propagated based on known dependencies.

With the sample database shown in Figures 1 and 3, Kaleidoscope takes a few seconds to update its menu state on XEROX 1186, which runs at about 0.75 MIPS. This update interval, however, can be reduced by a factor of ten or more if Kaleidoscope is ported to today's RISC-based workstations.

9 Relation to Other Work

9.1 Cooperative Response Systems

Past research in artificial intelligence proposed knowledge-based postquery cooperation to increase the usability of NLI systems. At the parsing level, one direction of research sought the system's robustness to extragrammatical sentences [Carbonell and Hayes, 1984]. At the conceptual level, following Grice's principle of cooperation [Grice, 1975], so-called cooperative response systems dealt with the user's misconception about underlying information systems [Corella *et al.*, 1984, Gal and Minker, 1987, Janas, 1981, Kaplan, 1982, Kao *et al.*, 1988, Mays, 1980, McCoy, 1985]. When queries fail to produce meaningful results because of the user's misconception, the system resolves specific causes of failure for the user. Yet, in these postquery cooperation approaches, the system still does not use its knowledge until the user query fails.

Kaleidoscope takes a more active attitude in utilizing the system's knowledge: a system knowledgeable enough to correct or to suggest the postquery correction should use its knowledge first to guide users away from query failure. The increasing speed of computers makes it feasible for the system to take this initiative. Nevertheless, postquery cooperative response would be still needed to handle queries that have no matching tuples in the database or produce too many or too small tuples. Even in such a case, the system's knowledge should be used actively. Consider an extensionally failing query with $\mathcal{F}$, a set of conjuncts causing the failure. Instead of just informing the user of $\mathcal{F}$, the system suggests alternatives in query generalization focusing on this set of literals. For instance, if the keyword specification of books belongs to $\mathcal{F}$, the system suggests its generalization based on the hierarchy of keyword values. Previous research explored a range of options for such query generalization [Motro, 1986, Chaudhuri, 1990].

9.2 Menu-Based NLI Approach

As windows and pointing devices such as the mouse become widely available for human-computer interaction, Tennant and Thompson recognized that the window-based interaction could restrict the users of so-called NLIs within the system's limited linguistic and conceptual coverage [Thompson *et al.*, 1983]. This idea has developed into so-called menu-based NLI systems NLMenu [Thompson *et al.*, 1983], INGLISH [Phillips and Nicholl, 1985], and NLParse/NLGen [Hemphill *et al.*, 1986]. A context-free, semantic grammar specifies dynamic choice generation in NLMenu and INGLISH. NLParse/NLGen employs a unification-based grammar to pursue linguistic generalization.

Kaleidoscope takes the notion of grammar-driven menu guidance from these menu-based NLI systems, and provides a model-based framework for the grammar design and interface generation. A semantically rich model provides the basis for user guidance and interface design. In contrast, the past research assumed a very low-level model or no explicit model at all. For example, the underlying model of NLMenu grammar is not much different from the relational model [Thompson *et al.*, 1987]. As a result, NLMenu queries are often reminiscent of formal queries. The emphasis on a model in Kaleidoscope also makes it possible to provide meaning-based guidance, which previous menu-based NLI systems overlooked.

10 Summary

Kaleidoscope provides an English-like query language for users to phrase queries with restricted yet common English expressions. A grammar-driven menu system bridges the inevitable mismatch between this language and the user's language. By generating legitimate EnQL constituents step by step as menu choices, this matching device relieves casual database users of learning and recalling the restrictions on EnQL and the specific concepts in a database. Users formulate queries in an English-like language by recognizing choices coming one after another that match their mental queries. The system uses its knowledge actively to guide users to create unambiguous and meaningful queries.

This paper has presented a formal data model for supporting EnQL, and a mapping to the relational storage. This model provides users with various degrees of freedom in query formulation. The presence of this data model is also very important for defining a transportable interface architecture. The model guides the design of a domain-independent grammar, a domain-independent query translator, and a set of procedures generating domain-specific lexicons from a schema.

To measure the gain in the user's benefit of using EnQL, we have relied on a syntactic measure - the number of tokens required to express a query. When SQL is taken as a reference, EnQL queries are significantly more concise than their SQL translations, often by an order of magnitude. In the future, we expect a human subject experiment to measure more semantic gains such as the user's conceptual freedom in expressing a query. We also expect future research to extend the expressive power of EnQL beyond conjunctive queries.

Acknowledgement

This research was supported in part by the DARPA contract N039-84-C-211 for Knowledge Based Management Systems. The authors wish to thank Prof. Terry Winograd, Dr. Charles Kellogg, and Prof. Jack Milton for their feedback on this research.

References

[Bobrow and Bates, 1982] Robert J. Bobrow and Madeleine Bates. Design dimensions for non-normative understanding systems. In *Proc. 22th Annual Meeting of ACL*, 1982.

[Carbonell and Hayes, 1984] J. G. Carbonell and P. J. Hayes. Recovery strategies for parsing extragrammatical language. Technical Report CMU-CS-84-107, Dept. of Computer Science, Carnegie-Mellon University, February 1984.

[Cha, 1991a] Sang K. Cha. Kaleidoscope: A cooperative menu-guided query interface (SQL version). *IEEE Trans. on Knowledge and Data Engineering*, 3(1):42–47, March 1991.

[Cha, 1991b] Sang K. Cha. Kaleidoscope: A model-based grammar-driven menu interface for databases. Ph.D. Thesis in preparation, Stanford University, 1991.

[Chaudhuri, 1990] Surajit Chaudhuri. Generalization and a framework for query modification. In *Proc. IEEE Data Engineering Conf.*, Feb 1990.

[Chen, 1976] Peter Chen. The Entity-Relationship Model - Toward a unified view of data. *ACM Trans. on Database Systems*, 1(1):9–36, 1976.

[Chen, 1980] Peter Chen. Entity-Relationship diagrams and English sentence structures. In Peter Chen, editor, *Int. Conf. on Entity-Relationship Approach to Systems Analysis and Design.* North-Holland Publishing Company, 1980.

[Corella *et al.*, 1984] Francisco Corella, S. J. Kaplan, G. Wiederhold, and L. Yesil. Cooperative responses to boolean queries. In *Proc. IEEE Data Engineering Conf.*, pages 77–93, April 1984.

[Forgy, 1981] C. L. Forgy. OPS5 user's manual. Technical Report CMU-CS-81-135, Dept. of Computer Science, Carnegie-Mellon University, 1981.

[Gal and Minker, 1987] Annie Gal and Jack Minker. Informative and cooperative answers in databases using integrity constraints. Technical Report CS-TR-1191, University of Maryland, September 1987.

[Grice, 1975] H. P. Grice. Logic and Conversation. In Donald Davidson and Gilbert Harman, editors, *The*

Logic of Grammar, pages 64–75. Dickinson Publishing Co, 1975.

[Hemphill *et al.*, 1986] Charles T. Hemphill, Inderjeet Mani, and Steven L. Bossie. Towards an effective natural language interface to knowledge based systems. Internal Working Paper, AI lab., Computer Science Center, Texas Instruments, Inc., Dallas, TX, December 1986.

[Janas, 1981] Jurgen M. Janas. On the feasibility of informative answers. In H. Gallaire, J. Minker, and J. M. Nicolas, editors, *Advances in Database Theory*, pages 397–414. Plenum Press, 1981.

[Jarke *et al.*, 1985] Matthias Jarke, Jon A. Turner, Edward A Stohr, Yannis Vassiliou, Norman H. White, and Ken Michielsen. A field evaluation of natural language for data retrieval. *IEEE Trans. Software Engineering*, SE-11(1):97–114, January 1985.

[Kao *et al.*, 1988] Mimi Kao, Nick Cercone, and Wo-Shun Luk. Providing quality responses with natural language interfaces: The null value problem. *IEEE Trans. on Software Engineering*, 14(7):959–984, July 1988.

[Kaplan, 1982] S. Jerrold Kaplan. Cooperative responses from a portable natural language query system. *Artificial Intelligence*, 19:165–187, October 1982.

[Kay, 1980] Martin Kay. Algorithm schemata and data structures in syntactic processing. Technical Report CSL-80-12, XEROX Palo Alto Research Center, October 1980. Also in B. Grosz, K. Jones, and B. Webber, editors, *Readings in Natural Language Processing*, Morgan Kaufmann Publishers, Inc. 1986.

[Marcus, 1982] Mitchell P. Marcus. Building nonnormative systems – The search for robustness. In *Proc. 20th Annual Meeting of ACL*, 1982.

[Mays, 1980] Eric Mays. Failures in natural language systems: Applications to data base query systems. In *Proc. AAAI*, pages 327–330, 1980.

[McCoy, 1985] Kathleen Filliben McCoy. *Correcting Object-Related Misconceptions*. PhD thesis, University of Pennsylvania, Dec 1985.

[Miller, 1956] George A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63(2):81–97, March 1956.

[Minker, 1988] Jack Minker, editor. *Foundations of Deductive Databases and Logic Porgramming*. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1988.

[Motro, 1986] Amihai Motro. Query generalization: A method for interpreting null answers. In Larry Kerschberg, editor, *Expert Database Systems: Proceedings from the First International Workshop*, pages 597–616. Benjamin/Cummings, 1986.

[Phillips and Nicholl, 1985] Brian Phillips and Sheldon Nicholl. INGLISH: A natural language interface. In *Foundation for Human-Computer Communication*. IFIP WG 2.6 Working Conference on The Future of Command Languages, 1985.

[Shneiderman, 1980] Ben Shneiderman. *Software Psychology: Human factors in Computer and Information Systems*. Winthrop Publishers, Inc., 1980.

[Simon, 1981] Herbert A. Simon. *The Sciences of the Artificial*. MIT Press, 2 edition, 1981.

[Stonebraker *et al.*, 1987] Michael Stonebraker, Eric Hanson, and Chin-Heng Hong. The design of the Postgress rules system. In *Proc. IEEE Data Engineering Conf.*, 1987.

[Sybase, 1988] Sybase, Inc. *TRANSACT-SQL manual*, 1988.

[Thompson *et al.*, 1983] C. W. Thompson, K. M. Roth, H. R. Tennant, and R. M. Saenz. Building usable menu-based natural language interface to databases. In *Proc. 9th Conf. on VLDB*, pages 43–55, 1983.

[Thompson *et al.*, 1987] C. Thompson, S. Corey, M. Rajinikanth, P. Bose, S. Martin, R. Roberts, R. Lewis, R. Enand, T. DiPesa, and S. Cha. RTMS: Toward close integration between database and application. In *Proc. of 20-th Annual Hawaii Int'l Conf. on System Sciences*, 1987.

[Widom and Finkelstein, 1989] Jennifer Widom and Sheldon J. Finkelstein. A syntax and semantics for set-oriented production rules in relational database systems. Technical report, IBM Almaden Research Center, 1989.

[Zdonik and Maier, 1990] Stanley B. Zdonik and David Maier, editors. *Reading in Object-oriented Database Systems*. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1990.