SMILE : A Visual Query Interface for an Engineering Database

Deepa Krishnan and Toshiyasu L. Kunii
Dept. of Information Science
The University of Tokyo
7-3-1 Hongo, Bunkyo-ku
Tokyo 113, Japan

Abstract

Querying and updating three-dimensional shape data in a
CAD environment is a complicated task. A new visual query
interface which allows a user to walk through a database,
climb up or down the database hierarchy or fly over the
database while picking, viewing and updating shape data, is
presented. No query syntax or schema details need be
memorized. Implementation using G-Base, a link-oriented
database management system is discussed and example
queries are illustrated.

1. Introduction

Graphical querying facilities reduce the cognitive load on
the user and have become an active research area since the
1980's. Most existing graphical database interfaces are
designed to handle only one-dimensional data and are
mostly schema-browsers. When the amount of data is
large, finding a specific object by browsing is often tedious.
Hence, in such a case, a query can be formulated to locate the
object precisely. It is important to allow a user to formulate
such a query with minimum effort. In most existing systems,
complex syntactical queries have to be formulated and keyed
in before their answers can be viewed on the screen. Further,
active memorized database knowledge is required. In
Computer-Aided Design (CAD) databases, the situation is
further complicated by the existence of hierarchically
organized machine parts and large amounts of design data. A
natural and user-friendly interface is urgently required.

A graphical query interface for a CAD application is
presented. The interface allows the user to traverse the
database naturally and pick, view and update data. Complex
nested queries are easily formulated. The user does not have
to be familiar with the semantics of the underlying model
whether conceptual or physical. All operations are performed
using a mouse and the entire environment is very interactive.

Section 2 discusses contemporary approaches to query
language interfaces. The querying formalism and its
operators are described in Section 3. Example queries are
then illustrated. Finally, advantages and extensions are
discussed.

2. A Survey of Existing Visual Query Interfaces

Recently, a number of query language interfaces have
been developed [1-5, 8-10, 12]. These interfaces are aimed
at simplifying the query formulation task and providing
visual answers. In this section, the approaches adopted are
discussed in detail.

The main approaches underlying contemporary visual
query language interfaces can be classified as given in Table
1. In approach A (See Table 1) graphics is used for schema
visualization and visual answers are provided for queries
which are syntactically formulated. This is better than an
absolutely naive approach but still does not simplify the
query formulation task. Systems, in which although queries
are syntactically formulated, knowledge of the underlying
model or methodology is essential, are also included in
approach A. Approach B is based on form-filling [4, 8, 12].
In QBE [12] for example, tables are displayed on the screen
and the user interactively constructs queries by specifying
examples of the desired answer set. Here, nested queries are
not easy to formulate and the answers are not graphical. Such
an approach deals mainly with 1-D data. G-Whiz [4], on the
other hand, is an extension of QBE to support nested
queries. The third approach C refers to the construction of
browsing interfaces. In this case, there is no formal querying
mechanism but the user is provided with adequate facilities
to explore and browse through the database [5]. Icon-based
interfaces are easy to use and involve a kind of procedural
language. However, there is no formalism involved. Lastly,
a hypertext approach [2] has been suggested. Though this
approach is suitable for browsing, it is a low-level approach
which cannot be used for formal querying and changes made
cannot ensure consistency. Many of the above systems are
based on the relational model and hence, relationships
between entities are not explicitly specified. For the user,
such relationships are hard to understand.

One interface that uses a graph-based approach is SNAP
[1], a comprehensive schema manager developed for the IFO
model. SNAP allows visual design and manipulation of
schemas. Restricted selection-type queries may also be
formulated. SNAP is comparatively easy to use and has a
number of visual functions [1]. Apart from those functions
that allow visual schema design, SMILE, the query language
described here, has most of the other functions for schema
manipulation that are a part of SNAP. In addition, SMILE
allows a user to browse through data without having to
formulate a query to do it. SMILE uses a different method
for query formulation which is more natural, intuitive and
easy to understand and use. In [1] it is unclear as to how
provides a formal querying mechanism for an object-oriented
query just from its visual representation [1]. In SMILE, the
formulated ones, while in SMILE G-Base queries can be
Further, it is difficult to understand the semantics of the
parametric shape model [6, 7, which allows high-level
D data. Visual interfaces for engineering databases have
representation of shape topology and geometry and has an
object-oriented flavour. This model is not designed here for
brevity. The user can walk through the database, moving
across links, climb up or down the schema hierarchy and
stop anywhere along his tracks to pick, view and make
changes. All changes are made consistent with the semantics
of the parametric and the graph data models. The user
communicates using a mouse and an intermediate procedural
language indicates the operations performed. The
corresponding query in G-BASE is easily derivable from this
intermediate language. No explicit method is provided to
construct schema-level queries but facilities are provided to
view the schema at various levels of detail. The user can also
alter the schema display by hiding or showing records and
links as required. The hierarchical structure simplifies the
understanding of complicated schemas and also enables the
user to view answers to queries at various levels of detail. Both
textual and graphical answers can be generated. The
interface provides a natural way of formulating queries and
performing update operations without unnecessarily
burdening the user with schema details or query syntax. No
a priori knowledge of the underlying model is necessary.
Furthermore, the graph data model provides the user with
simple record and link structures and no complex
relationship types are required to be understood or
memorized. Since logical access paths are explicitly specified
and each query is represented as a connected graph, the load
on the system is reduced and unambiguous queries are
formulated. SMILE is part of a browsing interface GRIN
described in [10, 11].

More details on the querying formalism and its associated
operators are given in the next section.

3. The Querying Formalism

The description of the query language is divided into two
parts. The first part discusses the schema structure and the
operators. This is followed by examples of different types of
queries.

The use of G-Base simplified the implementation of the
visual query interface to a large extent. This was due to a
number of useful features inherent in the Graph Data Model
(GDM), the underlying data model. Firstly, unlike in the
relational model, relationships are explicitly described using
links in GDM schemas. Thus, GDM schemas are easy to
understand. Further, these links, serve as paths for the user
to traverse and hence, facilitate the formulation of a high-
level navigational query language. A GDM schema
corresponds to a map and a user who has to go somewhere,
has just to choose the right paths to do so. Unlike languages
like SQL, the Graph Data Language (GDL) provides a record
list structure to store intermediate results of queries [1].
This is very useful in the query interface as intermediate
results can be maintained in record lists or targets as they are
called, and the user can carry these targets with him as he
continues on his journey through the database. Further, N-
ary links between record types can also be simulated by
using targets and n binary link types. Each query in GDM is
expressed as a connected graph. Since, logical access paths
are explicitly specified, the load on the system is reduced and
unambiguous queries can be formulated.

The DML of G-Base is relatively easy to understand
and queries can be easily formulated. G-Base offers an
environment to translate high-level semantics (as expressed
by the Entity-Relationship (E-R) model for example) into an
operational environment.

3.1 The Operators

The SMILE interface is navigational in nature and has
been designed to have two levels of operation. The first level
is designed for the novice user with little knowledge of target
definitions and path expressions. The user can move around
the database, passing by nodes and links retrieving whatever
is necessary using pre-defined targets. Updates and model-
specific operations can be performed using the routines in the
parametric model. At the second level, the more advanced
user can define new targets, perform projections and
additions on targets, use the count function, and define
complicated path structures.

A flat GDM schema for a certain application can have a
large number of nodes and arcs and can be very complex.
For example, the GDM schema of the parametric model
described in the previous section, has about 20 records and
45-50 links. It is difficult to show all this on the screen at one
time. Scrolling techniques could be used, but it is still

<table>
<thead>
<tr>
<th>Approach</th>
<th>Features</th>
<th>Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>graphics for schema visualization, symmetrically formulated queries, graphical or textual answers.</td>
<td>ENAM-based, LiD, GUIDE, GLAD and others</td>
</tr>
<tr>
<td>B</td>
<td>form-filling</td>
<td>Pasta-3, QBE, G-Whiz</td>
</tr>
<tr>
<td>C</td>
<td>Browsing interface, no formal querying</td>
<td>TWiBEER, SDDBMS</td>
</tr>
<tr>
<td>D</td>
<td>Icon-based, easy-to-use, procedural</td>
<td>IBS, CUPID</td>
</tr>
<tr>
<td>E</td>
<td>Hypertext-based, suitable for browsing</td>
<td>NEPTUNE</td>
</tr>
</tbody>
</table>

Table 1: Approaches to Query Language Interfaces
difficult for the user to understand the schema well by seeing portions of it at a time. Hence, a hierarchical structure is superimposed on a flat GDM schema as follows:

Let the GDM schema corresponding to a particular application be given as:

\[ S = (R, L) \]

where \( R = (R_i; i = 1,...,m) \) and \( L = (L_j; j = 1,...,n) \) and where \( m \) is the number of record types in \( R \) and \( n \) is the number of link types in \( S \). This schema can be expressed as a directed graph with record types as nodes and link types as edges. A semantically meaningful hierarchy is defined by dividing the schema into \( k \) levels. Each level is given as:

\[ \text{level}_q; q = 1,...,k = (VR, VL, T-links) \]

where \( VR \subset R \) are visible records i.e. records that can be seen and therefore accessed by the user at that level. \( VL \) denote visible links at a particular level and each occurrence of \( VL \) is given as \( L_j = (R_i, R_j) \) where \( R_i, R_j \in VR \). If \( LR_i \) denote all the links attached to \( R_i \), then \( L_j \subset LR_i \) and \( VL \subset L \).

T-links are tree links that connect one level to the next. A tree link is defined as:

\[ T-link = (R_i, R_{i+1}) \]

where \( R_i \in VR_i \) and \( R_{i+1} \in VR_{i+1} \). \( VR_i \) and \( VR_{i+1} \) are the visible records at levels \( i \) and \( i+1 \) respectively. A tree link connects a record at level \( i \) with its virtual image at level \( i+1 \).

These concepts are illustrated using PSM as an example. In defining a semantically meaningful hierarchy, visualizing the steps the user would take while accessing a database, based on this model, are considered. Typically, it may be assumed that he or she would first check the types of objects that exist in the database. Suppose the user then chooses the FEATURES level to operate in. First, he or she would like to see what features exist in the database. So the first level has only the record FEATURES. Having seen the features that exist, it is presumed now the user would like to know more about the data associated with a particular feature. He or she would then slide into the next level using the T-link that connects FEATURES with its virtual image at the second level. This is the template level and is different for a versioned or a non-versioned feature (Fig. 1). After examining this level, the user can now slide into the instance level to select a specific instance (Fig.1).

For the novice user, four types of graphical operators are defined. These are:

1. Traversal operators
2. Picking operators
3. Viewing operators
4. Auxiliary operators

Each type is described in detail.

Traversal Operators

**zoom_in**: This operator operates on two different targets. These are level and record. For these two versions, the formal characterizations are as follows:

**zoom_in (level)**:

\[ \text{level} = \text{level} + 1, \text{if current_entity}=\text{NULL} \]
\[ \text{level} = \text{level} + 1, \text{if current_entity}=\text{R}_i \]

**zoom_in (record)**:

\[ \text{current_entity}=\text{R}_i, \text{where R}_i \text{is the selected record.} \]

Here, current_entity denotes the simulated current position of the user in the database as he or she is moving around. The **zoom_in (level)** operator can be explained as follows: When no current entity has been selected by the user (he may considered to be outside all the nodes, or to be viewing the database aerially), the execution of the operator just causes the next level of the schema to be displayed. When a current entity exists, the operator causes the user to slide down to the next level using a tree link to land into the virtual image (denoted by \( VR_i \)) of the current entity at that level. The **zoom_in (record)** creates a new current entity. Similarly the formal characterization of the **zoom_out** operator is as follows:

**zoom_out (level)**:

\[ \text{level} = \text{level} - 1, \text{if current_entity}=\text{NULL} \]
\[ \text{level} = \text{level} - 1, \text{VR}_i \rightarrow \text{inverse T-Link}_i \rightarrow \text{R}_i, \text{if current_entity}=\text{R}_i \]

**zoom_out (record)**:

\[ \text{current_entity}=\text{NULL.} \]

This operator is clearly the inverse of the **zoom_in** operator. The level version of the **zoom** operators allow a user to travel through levels, i.e. up or down the database hierarchy. By using these operators in combination with the record versions, the user can travel through the levels while living inside a certain entity. The next two traversal operators help a user to travel within a level. These are the **walk** and the **return path** operators. Their formal characterizations are given below:

**walk**:

\[ \text{R}_i \rightarrow \text{L}_i \rightarrow \text{R}_j, \text{current_link}=\text{L}_j, \text{current_entity}=\text{R}_j \]

where \( \text{R}_i, \text{R}_j \in VR \) at the current level and \( \text{L}_i \in VL \) at the current level. The current entity is set to NULL indicating that the user is now positioned outside all entities.

**return path**:

\[ \text{R}_j \rightarrow \text{inverse L}_i \rightarrow \text{R}_i, \text{current_link}=\text{NULL}, \text{current_entity}=\text{R}_i \]

where \( \text{R}_j, \text{R}_i \in VR \) at the current level and \( \text{L}_i \in VL \) at the current level.

This is the inverse of the **walk** operator.

Viewing Operators

There is one viewing operator called **scan** which operates on schema, level, record, attribute, and instance. The **scan (schema)** operator displays the entire flat GDM schema on the screen and offers a simple means to serially view the entire database. The **scan level** operator, on the other hand, displays the schema at the current level. The action of the **scan record** operator is given as: for \( \text{R}_i \) print name, where \( \text{R}_i \) is the current entity. The **scan_attribute** operator enables
the user to see the attributes associated with a record. The last scan instance operator helps the user to see, in detail, the attribute values of an instance of a record type. Corresponding erase operators serve as inverses.

Picking Operators

The pick operator selects instances of the current entity which satisfy a predicate P. A formal characterization is as follows:

$$\text{pick: } \sigma_{r \in R, P(r)} R.$$  

The result is stored in a target. The inverse operator put clears the result of the most current pick.

Auxiliary Operators

There are three auxiliary operators. A print operator displays the answers to a query. The hide operator operates on three targets. These are records, links, and attributes. The selected record, link or attribute is hidden to make the display more presentable or for any other reason desired by the user. The inverse operator show also operates on the same targets and makes the target object visible on the display. The action of the hide operator on attributes is the same as the projection operator and the result obtained can be denoted as $\Pi_{\text{list}} (R)$, where the attributes in the list are picked using a mouse.

These operators are more than sufficient for the naive user to traverse the database and construct complex queries. Examples of queries are illustrated later in this section. Target-level queries can be formulated by advanced users. For details, refer [6].

3.2 Example Queries

A portion of the database is read into the GRIN interface before a design session [7]. SMILE allows queries to be formulated either on the data that has been read in or directly on the database. This is to increase the speed of response, and flexibility (the user can test a query on the data that has been read in before executing it on the database, minimizing errors). Answers to queries executed directly on the database can be textual (as when the DML is used) or graphical (the answer has to be read into GRIN before being displayed). Some examples of various types of intensional and extensional level queries are shown below.

Intensional Queries

In SMILE, the user does not formulate intensional level queries. These queries are simply answered by viewing the schema at various levels of detail. For example, a query such as "What are the attributes associated with the entity FEATURE?" can be answered by choosing the command zoom attrib from the menu and clicking on the entity FEATURE. Similarly, "What are the links associated with the entity INSTANCES?" is answered by first selecting the INSTANCES entity and zooming down one level in the schema. The same result can be achieved by first zooming down to the level where the details of this entity are displayed and then using the show command to view only the part of the schema concerned with the entity INSTANCES. The result is shown in Fig. 1.

The fact that there are only two types of links in G-Base (unlike other systems with so many complex relationships) allows the development of a simple browser facility to answer schema-level queries.

Extensional Queries

Extensional queries can be divided into simple attribute, instance-related, template-related, queries that use the parametric model routines and target queries. The intermediate dml indicates the commands chosen by the user to formulate his or her query. In other words, this corresponds to the path taken by the user while travelling through the database.

The answer to the query "Print the names of all the features in the database" in the graphical mode is shown in Fig. 1. The answer is obtained by selecting the entity FEATURES and choosing the scan record operator. Both simple and complex conditions can be defined (see Fig. 1). Complex conditions are used to express conjunctive, disjunctive, and negation queries.

Instance-related queries are also easily answered. The answer to the query "Which are the instances of shape V_BLOCK with height less than 100?", the definition of a condition, the intermediate dml and the derived G-base query are shown in Fig. 1. Similarly, queries related to templates can also be formulated. Other types of queries are those that use the viewing, object creation and manipulation routines in the parametric shape model [6, 7]. These routines have been designed to support the design task and allow objects representations to be manipulated naturally. Routines that allow the display of feature trees can be used to provide answers to recursive and transitive-closure queries [6].

Advanced target queries using aggregate functions (such as the COUNT function) can also be formulated by using only the mouse. Such queries are not illustrated here for brevity but the interested user can refer to [6].

4. Advantages

SMILE provides a natural way of travelling through the database and there is no need to remember schema details or query syntax. Queries are formulated with minimum effort and errors are rare. Partial or lazy-evaluation of queries is possible to check the results. All updates are consistent with the parametric model and can be verified. The browsing facilities of the main GRIN interface can be used in conjunction with SMILE for better results. Modular querying is facilitated. SMILE also offers two-level support for expert and novice users. Due to the one-to-one correspondence between the operators of SMILE and query expressions in G-Base [11], G-Base queries can be easily derived from the intermediate language.

One drawback is that long meaningless queries can be generated when the user repeatedly traverses and re-traverses paths in the schema. One way of preventing this is to get the user to turn-off the "build-query" option when he or she is just moving around and is not sure of what to do. A simple algorithm can be defined to eliminate such queries. The algorithm traverses the path taken by the user (while formulating a query) marking the nodes and links encountered. During query formulation, the operations performed at each node are recorded. When a cycle is encountered, the operations performed at each node in the cycle are checked. If no pick has been performed at any of the nodes or the effect of the pick has been cancelled by a put, the cycle is eliminated. The effect of update operations performed in the cycle are not cancelled by this elimination as the changes have already been recorded. This is repeated till all nodes in the user's path have been marked.
5. Conclusions

An interactive visual querying interface SMILE for a CAD application was presented. Together with the browsing and the animation facilities in the parent GRIN interface, SMILE offers a natural querying facility and a user-friendly environment that is unmatched in today's systems. The system is currently running on the Sun SPARC series of workstations and is being extended to incorporate more complex querying.

References