Definition of CQL, a Visual Query Language

Shi-Guang Ju and Sergio V. Chapa
Department of Computer Science, CINVESTAV-IPN
Mexico City, 07360, Mexico
jushig@delta.cs.cinvestav.mx
schapa@alpha.cs.cinvestav.mx

Abstract

We present a visual language, CQL, for geographic database retrieval. CQL introduces a new visual element, card, as its primitive pictorial element. It allows the end user to create his programs visually and to represent query sentence expression in full graphics. The syntax and semantic of CQL are described in this paper.

Key Words

Visual language, programming environment, icon, database

1 Introduction

The use of visual aids in man-machine interfaces is growing rapidly thanks to the ever-increasing availability of new low cost technology in the fields of graphics, image processing, video, and to expanding interest in multimedia communication. This has led to an increasing interest in the use of visual languages and visual interface tools.

Visual languages have been proposed (Papadias and Sellis, 1995) as a valuable interface tool, since icons (the primitive pictorial elements of visual languages) represent conceptual entities (Chang, 1987) directly, and operations on such entities may be represented by suitable arrangements of icons. The efficacy of visual languages is closely linked to the capability that icon images have of being intuitively understood by most users. As a result, the user can easily form an intelligible, consistent and cohesive conceptual model of the system.

The database field is one of the most promising application areas of visual languages for many reasons (Ju, 1998). One reason is that as the social demand for database use expands, many people have had the opportunity of manipulating database systems by themselves. The manipulation of database systems, however, may sometimes create problems for the user, especially for the occasional user who may be unfamiliar with sophisticated database manipulation or who does not know a programming language like SQL. Moreover, in comparison to using a restricted natural language, both the dependence on the native language of the user and the limitations imposed by the application area are avoided. Finally, the process of query formulation itself encourages the use of visual languages.

This work was supported by Jiangsu Natural Science Foundation, China under contract BK99110 and CONACYT, Mexico under contract Red-Informatica.
The query language Cigales, for example, allows users to draw a query (Calcinielli and Mainguenaud). Unlike spatial-query-by-sketch, Cigales requires the users, prior to drawing the sketch, to select the type of spatial relation they are addressing. For instance, to specify that the road enters the park, the user would have to select the “intersect” operation and then draw the particular configuration; this can lead to models interfaces which may be awkward and tedious.

In a similar attempt, Lee and Chin (Lee and Chin, 1995) designed an iconic query language in which users compose a query by selecting spatial relations from a predefined set represented as icons. They only consider a small subset of topological relations, which a user can select from a set of icons.

In the past, sketching was used in CAD primarily for design. Later, spatial constraints were used for describing consistency in spatial databases in (Hirata and Kato, 1993); however, instead of describing situations that should match the configuration of interest, they focused on constructing those situations that would establish unacceptable database states. Although the language was iconic rather than sketch-based, it shares similarity with the principles of sketching.

Spatial relations have been considered as a secondary criterion for image retrieval systems that focus on shape similarity (Del bimbo et. al., 1994). The shape measures are quantitative and thus expensive to process in a spatial database, and the spatial relations considered use rough approximations based on minimum bounding rectangles. In contrast, spatial-query-by-sketch prefers qualitative measures, starting with the spatial relations among the objects drawn, and resorts to quantitative methods only to prioritize hits.

Max (Egenhofer, 1997) designed a visual language, based on principles of spatial-query-by-sketch, for geographic information systems. Users interact with spatial-query-by-sketch by using a pen to draw an example of the configuration they are interested in. Spatial-query-by-sketch parses this graphical input, analyzes it and translates it into a database query.

The card query language (CQL) described and discussed here is designed for a geographical information system applied to petroleum exploration. The design of such query languages must tackle problems that are in many ways different from those raised in conventional query languages. First a query language in a GIS must be able to answer queries of spatial nature (Egenhofer, 1994), i.e., queries corresponding to spatial relations between objects and/or entities. Second, some formalisms use only lines or points as atomic objects, while ours admits almost arbitrary complex symbols. Third, several formalisms do not use attributes at all, while ours allows arbitrary complex attributes or, in some cases, even nested object structures. Finally, in many formalisms, relations are explicitly given as uninterpreted and abstract, whereas our methods use implicit relations that are expressed as operations between entities.

Due to space limitations, we divide the contents in two parts: the definition of CQL and applications of CQL. This paper gives the definition of CQL.

2 Card - Expansion of Icon

Symbolic images and related structures have been used in a number of visual programming applications (Ju, 1998). Symbolic arrays (hierarchical symbolic images) and symbolic spatial indexes (symbolic images where the symbols correspond to direction and to representative topological points) have been used in applications including context-based retrieval in image databases, spatial reasoning, path planning (Holmes and Jungert, 1992) and image similarity retrieval (Lee et al., 1992). A pictorial query-by-example (PQBE) language (Zloof, 1977) was designed for the retrieval of previous direction relations from symbolic images. Symbolic images are commonly known as icons. In Chang’s classification (Chang, 1987), an icon is a two-part representation of an object, an action or a computational process, written as (Xm, Xi). Xm is the logical part (the meaning), and Xi is the physical part (the image).

In CQL, more general primitive pictorial elements are envisaged as graphical query primitives having meaning, image, parameters and characteristics.

2.1 The Definition of Card

Definition 1: A primitive pictorial element that describes a primitive entity, an action or a computational process and is represented by a four-tuple, [Xm, Xi, Xt, Xp], where

- Xm is the meaning of the element,
- Xi is the image of the element,
- Xt is the type of the element, PROCESS or OBJECT,
- the distinction between an object and a process depending both on context and interpretation, and
- Xp is the parameter of the element.

is called a card.

We use cards for representing concrete objects in order to facilitate recognition of functional meaning, goodness of drawing, suitableness of the object and to carry a maximum of information. Generally, the recognized function of an object will depend on the environment in which the object is used. Therefore, cards must be designed with these considerations in mind.

In our case, where the database applies to petroleum exploration, the cards intend to represent three type of entities:

* The first type includes the real entities that have an inherent visual representation— pictorial objects that are associated with a certain logical interpretation. In the system, those entities are exactly the spatial data of database which are the real objects on the earth’s surface.
Some examples include highways, rivers, roads, oil-wells and houses.

* The second type consists of those real entities that do not have an inherent visual representation. These include traditional data types and application data types such as documents and databases. We call entities of this type conceptual objects. A card exists for each one in the database.

* The third type includes the entities that represent the actions or computational process of the system. In the query, this type corresponds to PROCESS-CARD, defined in the grammar G in the following section.

2.2 Designing the Appearance of the Card

We design the cards using the same form (Figure 1) for each of the three types of entities. Visually, on the face of a primitive card, there is a space Xm the name of the card, another space for Xp, the parameter that is indexed to the special object which is going to be used, and Xi, an area for drawings which visually represent one class of entities. It can be difficult and often unfruitful to find a universally accepted set of cards. Thus we allow cards to be user-defined, their shape tailored to the user’s particular needs and his own mental representation of the tasks and methods he wishes to perform.

We have mentioned that there are three types of cards in the system. For the cards for which Xt = PROCESS, we design the picture Xm according to the inherent meaning of the operation. For example, for the “OPEN” operation, we use a picture of a person opening a book, and the operation “INSIDE” is represented by one object inside another, with the picture of a person entering an area.

Conceptual object cards represent those files of the database which hold the relations between the entities. For example, the files “region”, “area”, “project”, “temporary object” are those which

manage information about relations between data files and about the structure of data files, as well as about the data files themselves. When the user constructs the database, the card generator tool asks for the drawing of each conceptual object. In this project, we designed the pictures according to the context of the files, enabling the end user to understand the card’s meaning as rapidly as possible. For example, we use a map of Mexico to represent the conceptual entity file “region”, and engineering maps to represent the file “project”. In our system, the files of conceptual object are managed using the B-tree data structure.

Another type of card represents real world objects, managed using multi-list data structures. These objects are the spatial objects inside the data files. For these, we let the user define the cards using the system’s card generator tool. We propose the symbols Xi for real objects, which conform to the standardized Mexican map-symbol system used by government map-making departments. This helps civil engineers to understand the functional meaning of the cards directly and easily.

It is not necessary for the users to know the system in detail, merely to know the meaning of each symbol. We use cards such as those shown in Figure 2 for the objects, conceptual objects and the actions of the database.

![Figure 1: A primitive card.](image)

![Figure 2: The left column gives some examples of conceptual object cards, the center column some real object cards and the right column some process cards.](image)
2.3 More About the Xp Parameter of the Card

The function of the parameter element Xp of a card is to represent an entity more precisely. A card can represent a set of entities, one entity, or a set of attributes of an entity, depending on the parameter used. In Figure 3(a), the card “OIL_WELL” without any parameter represents a set of the entity oil well. In Figure 3(b), the card “OIL_WELL” with the parameter “well-name” represents a set of the attributes “well_name” of the entity “oil well.” In the Figure 3(c), the card with parameter well-name=“Alendro” represents some entity whose name is “Alendro”.

![Figure 3](image)

Figure 3: A card with different parameters represents different sub-objects.

2.4 User-Defined Cards

There are three types of cards in the system. One of them is the card that is used to represent real geographical entities. In some applications, users need to define special cards. We offer a tool with which the user can design new cards for his application. The diagram of the tool is shown in the Figure 4.

![Figure 4](image)

Figure 4: Diagram of defining a new card by user.

3 Specifying the Card Query Language CQL

Many different formalisms for the specification of visual languages have been studied over the last two decades. Grammar-like formalisms range from early approaches like web and array grammars (Rosenfeld, 1976), and shape grammars (Gips, 1974) to recent formalisms like positional grammars (Costagliola et al., 1993), relation grammars (Ferrucci et al., 1994), unification grammars (Wittenburg and Weitzmann, 1990; Wittenburg, 1993), attributed multiset grammars (Golinand and Reiss, 1989), constraint multiset grammars (Marriott, 1994) and several types of graph grammars. There are also a variety of non grammar-like formalisms, including algebraic approaches and logic-based approaches (Helm and Marriott, 1991; Haarslev, 1995).

We propose a spatial symbolic combination formalism for the specification of CQL.

3.1 Definition of VQS

Definition 2: A structure that is formed by arranging cards spatially according to the syntax of CQL visually represents simple actions of a query; such a structure is called a Visual Query Sentence (VQS).

3.2 Definition of Card Query Language (CQL)

Definition 3: The CQL is specified by the triple [CD,G,B], where, CD is the card dictionary, G is a context-free grammar, and B is a domain-specific knowledge base.
Initially there is an element, $S \in N$, called the QUERY MACHINE symbolic image as described in the following figure.

Cards, which represent database entities about which queries will be made, are inserted into BOX, and BOX$_2$. Cards in BOX$_2$ specify criteria limiting the objects to be searched for as described by the cards in BOX$_1$. Cards in BOX$_1$ define operations and relations between cards in BOX$_1$ and BOX$_2$, thus determining a query action.

$P$ is a finite set of productions or replacement rules. Each production in $P$ has the form:

$$\Gamma \rightarrow \text{CARD } \psi \text{ CARD } \ldots \psi \text{ CARD } , \Delta,$$

where

"$\rightarrow$" is read "can be replaced by". The meaning is analogous to the meaning of the card symbolic images when used in a conventional grammar. In contrast to the rules in a string language, which specify only the exchange of one substring for another, these rules must describe the exchange of one connected sub-structure for another.

CARD represents a card symbolic image.

$\psi$ represents compound relations among cards.

$\Delta$ is the rule for the production. There are three rules for production $P$:

- $p_1$: CARD $\rightarrow$ CARD + new attribute
- $p_2$: CARD $\rightarrow$ CARD + parameter
- $p_3$: BOX$_1$ $\rightarrow$ BOX$_1$ + CARD

An example of the product $p_3$:

The information given in the rule is as follows:

$\Gamma$ is a nonterminal symbol, each CARD is a grammar CARD, and each $\psi$ is a compound relation of the form

$$\psi = (R^1, \ldots, R^{k-1}, R^k, R^{k+1}, \ldots, R^n)$$

Each $R^k$ denotes a pair $(R, k)$ where $R$ is a relation in $Q$ relating the values of the syntactic attributes of CARD with those of CARD, for $1 \leq k \leq n$, $1 \leq i \leq n$.

$Q$ represents the relation identifier between the cards.

$E$ is a pictorial parameter evaluator which converts visual representations into numeric elements. Each symbol of the grammar has associated attributes, named syntactic attributes, to express additional syntactic information about the symbols (the card parameters). For example, given a card symbolic image, the coordinate pair at the centroid of the card on the screen is a syntactic attribute.

The grammar $G$ determines how many complex structures or VQSs can be constructed by spatially arranging elementary cards. Usually, nonterminal symbols in $G$ besides the start card QUERY MACHINE represent composite cards, that is, they can be derived in one or more steps, starting from a given nonterminal N which is different from S.

In other words, composite cards are obtained by spatial arrangement of elementary cards. A visual query sentence VQS contain at least one process card with at one object card.

### 3.3 Diagram of the Syntax of the CQL

We give the diagram of the syntax of the textual language corresponding to the CQL as follows. Words in capital letters represent a card symbolic image.
Definition of CQL, a Visual Query Language

- **Card**
  - User-defined object card
  - System-generated temporary object card

- **Process card**
  - Table operation
  - Orientation operation
  - Metric operation
  - Topographic operation
  - System operation

- **Table operation**
  - Group by
  - Order by
  - Union
  - Intersect
  - Minus

- **Orientation operation**
  - Above
  - Below
  - To right of
  - To left of
  - North of
  - South of
  - East of
  - West of

- **Metric operation**
  - Near to
  - Far from

- **Topographic operation**
  - Inside
  - Outside
  - Alongside of
3.4 An Example of Program with CQL

Figure 5: An example of VQS
Figure 5 shows a program in CQL. In this case, the program consists only of a VQS. There are 7 cards in the VQS. The QUERY MACHINE consists of three box cards. In BOX1, there are two real object cards RIVER and OIL_WELL which came from the real object library with numbers 1_4 and 1_7 respectively. In BOX2, there is a conceptual card PROJECT that came from the conceptual object library with number 2_5. A card INSIDE, from the process card library with number 3_4, is on the face of BOX3.

Two tables are used to represent E and Q:

**E:**

<table>
<thead>
<tr>
<th>Card Number</th>
<th>Card Name</th>
<th>Card Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>1_4</td>
<td>OIL_WELL</td>
<td>null</td>
</tr>
<tr>
<td>1_7</td>
<td>RIVER</td>
<td>null</td>
</tr>
<tr>
<td>3_4</td>
<td>INSIDE</td>
<td>null</td>
</tr>
<tr>
<td>2_5</td>
<td>PROJECT</td>
<td>proj_name=DETALLE_DR_COSS</td>
</tr>
</tbody>
</table>

**Q:**

<table>
<thead>
<tr>
<th>Card-Num</th>
<th>box1</th>
<th>box2</th>
<th>box3</th>
<th>1_4</th>
<th>1_7</th>
<th>3_4</th>
<th>2_5</th>
</tr>
</thead>
<tbody>
<tr>
<td>box1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>box2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>box3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1_4</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1_7</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3_4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2_5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

$\Psi_1: (0,0,0,1,1,0,0)$

$\Psi_2: (0,0,0,0,0,0,1)$

$\Psi_3: (0,0,0,0,1,0)$

$\Psi_4: (1,0,0,0,0,0)$

$\Psi_5: (1,0,0,0,0,0)$

$\Psi_6: (0,1,1,1,0,1)$

$\Psi_7: (0,1,0,0,0,0)$

0 represents the NULL relationship.

1 represents a relationship.

3.5 Semantics of the Operators of the CQL

In this section, we describe the syntax and semantics of the process cards. Due to space limitations, we only give a few examples. In all examples, the following notation is used:

- f: flat file,
- f(C): the file defined by the card C in a VQS,
- E(f,s): to take out the entities s from the file f.

### 3.5.1 Card OPEN

**syntax:**

![Figure 6: The syntax of the card OPEN](image)

**semantics:** The card OPEN represents an operation that opens an entity as a flat file and displays all the entities of the file on the screen.

**example:** In BOX2 of the VQS shown in Figure 6, there is a conceptual card "project" with parameter, proj_name="DETALLE_DR_COSS" which represents a file of the database. In BOX3, there is a card "OPEN" that represents an action that will open the file "DETALLE_DR_COSS" as a flat file.

**processing algorithm:**

- input: VQS.
- output: Visual result, or error-messages.
- processing: parse VQS, and interpret it to operation-lines:
  - interpret header of file f;
  - while there are more entities s in the file begin
    - next E(f, s);
    - explain the spatial data of E(f, s);
    - put out the E(f, s) to the output-lines;
  - end

### 3.5.2 Card ABOVE
3.5.3 Card INSIDE

To retrieve the entities S1, which are defined in BOX, and are located inside the entity S2 that is defined in BOX_a.

<example> The VQS shown in Figure 8 represents retrieval of the oil wells located inside the area "DETALLE, DR, COSS".

<processing algorithm>
input: VQS
output: 4 pipe-lines (graphics, text, temporal object, all) or error messages;
processing: parse VQS and interpret it to operation-lines;
interpret header of file f;
while there are more entities s in the file
begin
next E(f, s);
if test E(f, s) is above T1 then insert E(f, s) into the output-line;
end
To explain and put out the pipe-lines.

Figure 7: The syntax of the card ABOVE

Figure 8: The syntax of the card INSIDE
3.6 Card TEMPORARY_OBJECT

The temporary_object card is a special conceptual object card provided by the CQL to represent the intermediate results of a subquery.

Suppose that Y is a complex query. To obtain the query it is necessary to do n subqueries of the form:
\[ Y = f(x) = f(f^{-1}(f(y))) \]
where f(x) is the result of each subquery.

The subquery can be represented separately in the following form:
\[ y_1 = f(x) \]
\[ y_2 = f(y_1) \]
\[ y_3 = f^{-1}(y_2) \]
\[ Y = s = f(y_3) \]

CQL uses the TEMPORARY_OBJECT card to represent the intermediate results \( y, y_1, y_2, \ldots, y_n \) and \( \text{VQS}_1, \text{VQS}_2, \ldots, \text{VQS}_n \) for constructing n subqueries as in the following:

\[ \text{VQS}_1: f(x) \rightarrow \text{TEMPORARY_OBJECT} \]
\[ \text{VQS}_2: f(\text{TEMPORARY_OBJECT}) \rightarrow \text{TEMPORARY_OBJECT} \]

3.7 An Example Using the TEMPORARY_OBJECT Card

We have a query which would be described in natural language as to retrieve the names of all oil wells which were constructed after some specific oil well situated inside the project "DETALLE_DR_COSS", and were constructed in 1985.

The query can be expressed in SQL as:

```
select well_name
from oil_well
where date > some
    (select date
     from oil_well
     where proj_name = 'detalle_dr_coss'
     AND date > '841231'
     AND date < '860101')
```

The query will generate the set of date that satisfies the conditions that are defined in the clause where. We create a CQL program for this query as shown in Figure 9.

Figure 9: To retrieve the names of all oil wells which were constructed after some specific oil well situated inside the project "DETALLE_DR_COSS" and were constructed in 1985.
Conclusion

CQL introduces the card as the basic program element. It allows the end user to create programs visually, and represents sentence expression in full graphics. The semantics of an end user query are expressed by a combination of cards. The proposed CQL provides the following advantages for end users:

1. The end user can easily determine the target of queries from the card image Xi.
2. A civil engineer can easily create a query program with CQL.
3. The language proposes a new algorithm — temporary object — for sub-queries.

We describe how to create the program and applications for CQL in a subsequent paper (Ju and Chapa).

References

Egenhofer M., "Spatial SQL: A query and presentation language", IEEE Transactions on Knowledge and Data Engineering, 1994, 6, pp 86-95.
Ju Shi-Guang and Sergio V. Chapa, Application of CQL, a visual language. For publication.