Combining Constraints and Data-Flow in a Visual Query Language

Manoj Chavda
Department of Computer Science
University of Cape Town
Rondebosch 7700, South Africa

Peter T. Wood
Department of Computer Science
University of Cape Town
Rondebosch 7700, South Africa

Abstract
We define a graph-based visual query language for object databases, called QUIVER. Most graph-based query languages proposed in the literature view queries either as constraint (pattern) graphs or as data-flow graphs. QUIVER, however, allows constraints and data-flow to be combined in queries. This combination of constraints and data-flow is motivated by the need to support comprehensive object models, such as that proposed by the Object Data Management Group (ODMG). QUIVER supports all ODMG object model constructs, including computational constructs such as operations (methods). In addition, QUIVER supports computational query constructs such as subqueries and (aggregate) functions.

1 Introduction
Visual query languages for databases have been a topic of research for many years, a comprehensive survey being given in [1]. In this paper, we describe a graph-based visual query language for object databases called QUIVER (an acronym for “QUerying in an Interactive Visual Environment”), which allows constraints and data-flow to be combined in queries.

QUIVER is part of a larger project to provide a consistent environment in which to visualize object database schemas, queries and instances. With this approach, a schema graph can be viewed as a template for producing instance graphs, while a query graph can be viewed as a set of constraint subgraphs and a set of computation subgraphs. The constraint subgraphs represent the structural or value constraints which the query should satisfy with respect to the instance being queried, while the computation subgraphs represent the flow of data among computations.

The object model which QUIVER supports is that proposed by the Object Data Management Group (ODMG) [2]. In the current implementation, QUIVER queries are translated into the Object Query Language (OQL) proposed by the ODMG.

In the next section, we introduce the QUIVER notation for ODMG schema graphs. This notation is a subset of that used for QUIVER queries which are described in Section 3. Finally, we provide some concluding comments.

2 Schema Graphs
An ODMG schema can be represented as a directed, labeled graph. Each node and arc has an associated visual representation and denotes an ODMG object construct. As an example, assume that we have a database representing university data. Objects of type Student and Course are defined in the schema, each with an associated extent (the collection of all objects of that type). Object types are depicted as shaded circles, while extents are represented as rounded rectangles (called blobs) enclosing an appropriate object node. Literals are represented as unshaded circles. Examples of these representations can be seen in Figure 1, although the figure represents a query rather than a schema.

Characteristics of objects (which include properties and operations) are represented by labeled arcs. For example, a Student object has a property takes which has as value a collection of Course objects (once again represented by a blob in Figure 1).

3 QUIVER Query Graphs
In this section we describe QUIVER query graphs by example. A more formal and complete presentation is given in [3]. The example queries are formulated on the university database described above.

A QUIVER query \( Q \) to find those students who take only courses offered by the department with name “CS” is shown in Figure 1. Query \( Q \) includes four items which would not be found in a schema graph. The first is the subquery \( Q' \) which is the portion of \( Q \) enclosed by the rectangle in Figure 1. The second is the data-flow arc connecting \( Q' \) to its output node. The third is the subset constraint which is represented as a broad, unlabeled arc. The fourth is the use of boldface to indicate output.

The non-bold portion of \( Q' \) can be interpreted as a constraint to be satisfied by the object database. We want to find those course objects in the Courses extent which are offered by a department whose name is CS. For each such course...
match, the bold object node is made identical to the corresponding course node. The bold blob indicates that all such matching objects are to form a collection which is the output of the subquery.

The subgraph \( Q^{10} \) in the lower half of Figure 1 represents another constraint subgraph. We require that any student appearing in the output of \( Q \) must take a collection of courses which is a subset of the collection of courses found by \( Q^{10} \).

Cycles are permitted in both the constraint and data-flow subgraphs of a query. An example of a cycle in a data-flow subgraph is shown in Figure 2. Such a cycle does not give rise to a recursive computation, but rather asserts that the data found and/or computed represents a fixed point. The \( \text{bag()} \) function in Figure 2 takes as input the object named \( \text{Jones} \), and outputs a bag containing \( \text{Jones} \). The \( \text{element()} \) function performs the inverse computation, which means that the query will succeed.

One of the goals of QUIVER notation is that it should be consistent; similar concepts should be represented by similar visual constructs. For example, all computation nodes (which include operation, function and query nodes) are represented as rectangles. The rectangle representing a function node is filled in, since the computation is hidden from the user. On the other hand, a query node is represented as a rectangle which includes the nodes and arcs specifying the computation (see Figure 1). For the outer query, this bounding rectangle is not necessary and has been omitted in our examples. A function node has data-flow arcs entering and leaving it. Queries do not require an incoming data-flow arc as their input is implicitly the entire database.

As can be seen from Figure 1, QUIVER supports objects, literals, and collection types, as well as attributes, relationships and subqueries. Collection types include sets, bags, arrays and lists. In addition, structures, operations and (aggregate) functions are supported. A one-to-one correspondence exists between these ODMG concepts and the visual primitives. Hence, while languages such as GraphLog [4], GOOD [6] and Hyperlog [7] advocate a minimal set of visual constructs, a distinguishing feature of QUIVER is the richness of its visual constructs.

Another distinguishing feature of QUIVER is its minimal use of text. The use of text is limited to names defined in the schema being queried and to values of literals. In particular, there is no need to use variables in QUIVER queries, nor is output identified textually.

4 Conclusion

We have described QUIVER, a graph-based query language for ODMG-compliant object databases which combines constraints and data-flow computation. QUIVER is comprehensive in the sense that it includes visual representations for a large number of the modeling constructs found in object databases. This is in contrast to many other visual query languages which have concentrated on using a minimal number of representations. At the same time, a serious attempt has been made to provide a consistent organisation of the QUIVER visual representations.

In the current implementation, QUIVER queries are translated to ODMG QQL, after which they are evaluated by the O2 database system [5]. A full description of the translation algorithm is given in [3]. Based on the representation of ODMG databases as graphs as described in Section 2, we are reformulating QUIVER query evaluation in terms of an algorithm which alternates between graph pattern matching and data-flow computation.

In order to compare the usability of QUIVER and OQL, we conducted a limited user evaluation, involving thirteen fourth-year students. The results of the evaluation suggested that it is easier to construct correct queries with QUIVER than it is with OQL. This may be partly because keeping track of long paths of references and the ranges of variables is difficult in OQL. In QUIVER, on the other hand, the paths are visual and there are no variables. Queries which involve nested subqueries may also benefit from a visual representation.

Acknowledgements

We are grateful to O2 Technology for the use of O2 during the development of QUIVER, and to the Foundation for Research Development for financial support.

References


