Towards a *Unified Query-By-Example* (UQBE): UML as a basis for a generic graphical query language.

Miguel Angel Sicilia Urbán
Dpto. Informática
Univ. Carlos III (Madrid)
msicilia@inf.uc3m.es

Elena García Barriocanal
Dpto. CC. Computación.
Univ. Alcalá (Madrid)
elena.garciab@uah.es

Juan Manuel Dodero Beardo
Dpto. Informática
Univ. Carlos III (Madrid)
dodero@inf.uc3m.es

Abstract. A generic graphical query language for ODMG-compliant object databases is proposed, based on the ideas of *Query-By-Example*, and using UML-like diagrams as schema notation. Ease of learning for users coming from the relational world and support for non object-oriented data sources are also considered as design goals. The overall layout of the query language is described, illustrating its potential expressive power via ODMG OQL comparisons.
Keywords. QBE, graphical query language, ODMG OQL, object database queries, UML.

1. UQBE: a simple graphical query language for object databases.

Query-By-Example (QBE) is a graphical query language for relational databases developed decades ago at IBM by Moshe Zloof [1][2]. QBE is a complete domain calculus language [3] and its expressive power is proved to be equivalent to that of SQL [4]. Although it has been included in a variety of commercial products, ranging from mainframe character-terminal programs (like IBM OS/390 *Query Management Facility* [5]) to personal database engines (like Borland's Paradox [6]), it was originally designed to be used "sitting at a terminal". Modern *Visual Query Languages* (like Inprise *Visual Query Builder*, included in Borland Delphi) can also be considered as descendants of the QBE philosophy, but they're adapted to modern workstation user interfaces. In this position paper we present our initial results in the effort of designing a graphical query language for object databases, called *Unified Query By Example* (UQBE), based on the underlying ideas of QBE, but designed for a modern graphical user interface (GUI). Unlike other QBE-inspired languages for the object world, like OOQBE [7], we have designed a *generic* language, capable of supporting non object-oriented data sources as well (we do not cover this point here). UQBE design goals are the following: (a) standards-based, (b) simple and intuitive (the way QBE is), (c) strongly GUI-based and (d) easy to learn for users coming from the relational world and capable of supporting relational sources.

1.1. Related standards.

QBE queries are constructed by filling *table skeletons* extracted from a relational database schema. We have selected a similar approach by showing a class diagram, extracted from the object database schema, and letting the user fill some information to build his/her query. Given that the *Unified Modelling Language* (UML) is becoming mainstream in the software industry, we have selected UML-like diagram elements as the UQBE notation, trying to respect UML notation [8] as much as possible, but adding the needed user interface elements. Other hand, ODMG 2 *Object Query Language* (OQL), a textual language based on SQL92 among other influences [9], is the only non-vendor-specific query language for object databases. The work presented here only deals with UQBE definition, ignoring implementation issues and therefore, we have only aimed to translate UQBE queries to OQL text that can be used with commercial object databases[1]. From here, we assume that we are making queries on a database that supports ODMG 2.0 (or any equivalent model), so we'll use ODMG Object Model concepts (refer to chapter 2 of [9]).

1.2. UQBE graphical interface

User interface design for UQBE is a mixture of QBE and existing modelling tools like *Rational Rose*. In fact, UQBE would ideally be packaged with a modelling tool capable of importing object database schema. Figure 1 sketches UQBE overall screen layout.

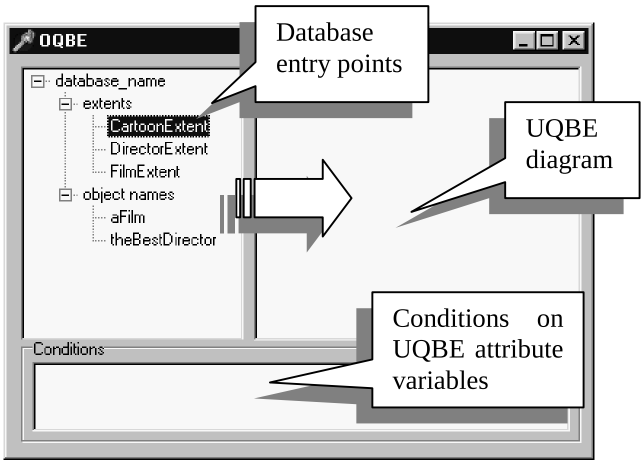

Figure 1. UQBE interface layout

A tree view on the left side shows the available database-scope **navigation entries** for the database, namely, extents and object names. The user opens the schema and can

[1] Due to the generic nature of UQBE, we also support other kinds of data sources through the use of Microsoft's ActiveX Data Objects (ADO) interfaces.

drag-and-drop some of them from the tree-view to the right pane, originating UML-like class boxes to be displayed, along with its associations.
Object names can refer to any single object, literal, or collection of any of them contained in the database. We indicate the kind (single element or collection) of each entry point with two UML stereotypes: <<*collection*>> or <<*singleton*>>, since queries are different for any of them (of course, extents are always collections). Class boxes represent the class of the single object or the base class of the collection (collection type is irrelevant in declarative languages). Attributes of object type are showed via UML associations with the corresponding type, and attributes of literal type are showed as UML attributes, in order to draw a clear distinction between querying elements inside an object or elements reachable by link traversal. The user can type expressions on the "conditions" window regarding schema elements. Finally, the result of a query can be an object or literal[2], or a collection of them, which are showed in a separate window[3].

2. Simple queries.

Let's first describe queries that don't involve navigation traversal. On a class box:

- we can mark an attribute's checkbox to select it (equivalent to '*P.*' QBE command), or the class' checkbox to show all the attributes. Note that checking the class returns a collection of database objects, while checking attributes will give as a result a query-generated collection.
- we can type comparison expressions following attribute names. The type of the attributes is showed in the tree-view or via *tool tips*[4].
- We can specify collating order by typing numbers in "*spin edit"* controls and selecting order from a *combo box* (ASC, DESC or none).

The results of the query will show the selected attributes of the object instances that satisfied the specified criteria, with a tabular representation. Given a database with an extent on class *Cartoon*[5] (see Figure 2), we could select attribute *title*, type a condition on attribute *year* to obtain films after between 1950 and 1980 and another on *title* to obtain films whose title begin with "An", and select primary and secondary sorting criteria. Generated OQL is the following:

```
select x.title
from   CartoonExtent x
where  x.year > 1950 and x.year < 1980
       and x.title like "An*"
order by x.year DESC, x.title
```

[2] An ODMG literal differs from an object in that it has not object identifier (OID); we only deal with atomic literals, and not collection or structured ones.
[3] Based on commercial tools like ObjectStore's *Object Inspector* or *POET Developer* that provide a spreadsheet-like navigable interface for instances.
[4] Small labels that appear when mouse pointer is paused on a user interface element.
[5] A sample database extracted from POET 6 OODBMS Trial Edition.

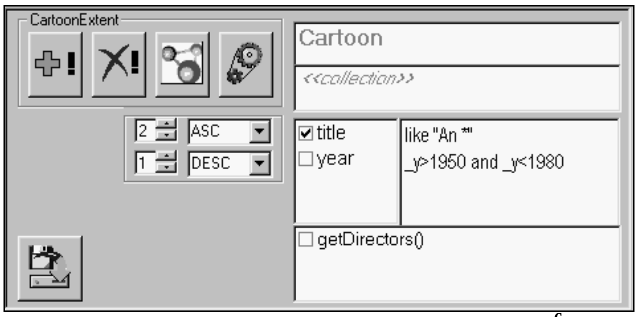

Figure 2. An UQBE query on a single class[6].

We'll obtain a collection of string literals as result.
We can also define variables by typing identifiers on the attributes (like `_y` in Figure 2), and we can use them in conditions. Conditions that affect multiple attributes of a class must be typed in a separate "*Conditions*" *list-box*, as in QBE conditions table. The semantics are the same as QBE *domain variables*, and we have used the same notation: the identifiers must start with an underscore. We can also put one of the OQL aggregation operators (*count*, *sum*, *min*, *max*, *avg*) on an attribute to obtain a single literal with the corresponding summary data.

3. Navigating associations.

Associations between classes are showed by default in the UQBE diagram. Since we execute queries only on entry point elements, other classes reachable form it via association traversal are automatically included (if the user marked something on them). To form the OQL query, we only need to build path expressions from that class to any other class that is selected in the query, using schema information. Figure 3 shows an UQBE query equivalent to the following OQL sentence:

```
select film.title
from FilmExtent AS film, film.directors AS
director
where director.Name = "Avery, Tex"
```

4. Additional features.

4.1. Schema and instance update

Inserting an object in QBE is as simple as writing values on the corresponding class box and clicking on the insert button (this works also for association implicit collections, and can be translated to construction OQL expressions). In a similar way, we can delete instances on an entry point by specifying a query and clicking delete button (instance deletion doesn't have a direct OQL translation). We have decided to implement updates on individual objects in the result window. Note that OQL lefts instance update to be done via method execution.
We have left unimplemented schema definition and update, since these manipulations are better handled from a separate modelling tool.

[6] Note insert, delete, filter and run buttons in the upper left corner and "save query" button in the bottom.

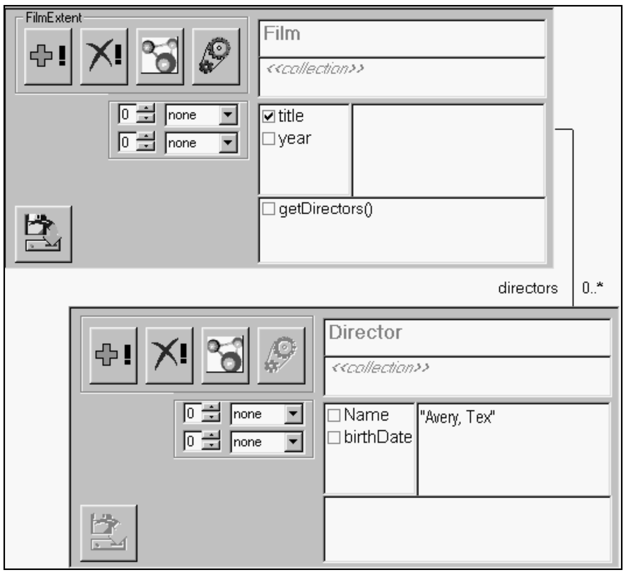

Figure 3. An UQBE **query involving association traversal 4.2. Nested queries and method invocation.**

To keep UQBE diagrams simple we have adopted a compositional approach for nested queries. UQBE allows the user to save a query and associate a name with it (these named queries are showed in the UQBE *tree view*). After that, the user can *drag and drop* that queries on the right pane, and a class box will show representing the results of the query (single instance or collection). The user can then make a query on the result of a previous one, just like he/she does on an entry point. The result is a nested OQL query. Method invocation can be used in a similar way. Method invocations that return a collection or single instance can be saved as a named query, and additional queries can be defined on them. The important point is that method invocation result and query execution are treated exactly as entry points, since they ultimately are defined from one of them.

4.3. Filtering collections.

Any UQBE collection defined on a base class C can also hold instances of any subclass of C. Filtering (sometimes called dowcasting, as in [10]) is a very common operation on collections that returns a new collection, with base class one of the subclasses of the original collection.
Filtering is implemented in UQBE through hierarchy buttons on collection class boxes. When we click one of these buttons, a separate windows shows us the inheritance hierarchy of the base class of the collection, letting us select one of the classes (the filtering class, that appear as a tool tip on the name of the extent). Actual filtering is done by the UQBE interface, and it's not specified in OQL.

5. Conclusions.

Our work establishes the basis for a simple graphical query language based on UML that follows a very different approach to object visual queries than the one provided in languages that use a 3D virtual interface (a relevant example is [11]) or a graph-based complex notation, like Quiver [12]. UQBE is simpler, easier to implement, and better suited for integration with existing modelling or database administration tools.

In our first prototype, developed on Windows platform, we have only covered a small portion of the complex OQL syntax[7] (we use POET OQL to test our prototypes, and it only supports ODMG OQL *select* clause), and therefore much work remains to be done. But we believe that a restricted implementation like the one described here brings some important benefits for the object database market:

- UQBE is a good choice for simple interactive queries, or whenever the user is novice or comes from the relational world.
- UQBE can be used as a good teaching aid for object database or object modelling basics.
- Object database and modelling tool vendors would be able to offer a common visual query language, improving their user and administration suites.

A simplified version of UQBE can be used for relational databases, making joins between collections by marking the foreign key of a table (represented as a class) and the key in the parent table with the same attribute variable (just like in QBE). This generic design follows the spirit of some "relational-compatible" ODMG constructs like *tables, keys* and *joins*, and facilitates transition from the relational paradigm.

6. References

[1] Zloof , M., "Query By Example", NCC, AFIPS, 44, 1975.
[2] Zloof , M., "QBE: A Language for Office and Business Automation", Computer, pp.13-22, May 1981.
[3] Ullman, J.D, Principles of databases systems, Second Edition, pp. 207-209, Computer Science Press, 1982.
[4] McLeod, D., "The translation and compatibility of SEQUEL and Query By Example", Proc. Intl. Conf. Software Engineering, San Francisco, CA, 1976.
[5] IBM, "Using QMF Version 3 Release 3", Document Number SC26-8078-01", April 1997, Appendix 1.1.10.
[6] Prestwood, M. A., Corel Paradox 9 Power Programming, Osborne/McGraw-Hill, August 1999, chapter 11.
[7] Staes F., and L. Tarantino, "OOQBE*: An Intuitive Graphical Query Language with Recursion", Human-Computer Interaction: Software and Hardware Interfaces, Vol. 19B, (Salvendy, G., ed.), Elsevier, 1993.
[8] Object Management Group (OMG), "Unified Modelling Language Specification", Version 1.3, June 1999, available at www.omg.org.
[9] Cattell, R. (Editor) *et al*, The Object Database Standard: ODMG 2.0, Morgan Kaufmann Series in Data Management Systems, Morgan Kaufmann Publishers, June 1997.
[10] Blaha, M. and Premerlani, W., Object-Oriented Modeling and Design for Database Applications, Prentice Hall, 1997, p.101.
[11] Murray,N. "Kaleidoquery: A Visual Query Language for Object Databases", Proc. of Advanced Visual Interfaces, L'Aquila, Italy, May, 25-27, 1998.
[12] Chavda, M., Wood, Peter T., "Towards an ODMG-compliant Visual Object Query Language", Proc. 23th Intl. Conf. on Very Large Databases, Athens, Greece, 1997.

[7] In addition, ODMG 3 standard has been recently published.