GOOD: A Graph-Oriented Object Database System

Marc Gemis Jan Paredaens
Inge Thyssens Jan Van den Bussche*
University of Antwerp (UIA)†

1 Introduction

In this video session we demonstrate a graph-oriented database management system, called GOOD [1]. The scheme of a database is represented as a directed graph. Also the database instance is (conceptually) represented as a graph. However, such an instance graph contains all information stored in the database and is therefore too complicated to be displayed completely on the computer screen, in a user-friendly way. It would be almost impossible to find the desired information, not to mention how difficult it would be to make directly changes in such a graph. Therefore we developed a language that simplifies the information retrieval and modification.

2 The language of GOOD

A formal definition of the GOOD-language can be found in [2, 3]. The language has only five basic graph transformation operations (node and edge additions and deletions and a duplicate eliminator) which can be combined into program and method constructions. With such programs we can express database queries, updates, scheme constructions and restructurings in a uniform way, as explained in [4]. GOOD can account for a number of object-oriented features, as shown in [5].

All operations are based on pattern matching. This means that the parts of the database on which the operation must work, are described by means of a graph pattern. The operation will be applied for every matching of this pattern in the database graph. Patterns in isolation have the power of select-project-join queries; the operational power of full language is computationally complete, as shown in [5, 6]. To our knowledge, this is the first graphical user interface with this property.

Syntactically, each operation is a graph, consisting of a pattern with special nodes and edges to indicate which nodes or edges have to be added or deleted or where duplicates have to be eliminated.

The base language has been enriched with macros, which are formally defined in terms of the basic operations in [5, 7]. The purpose of these macros is twofold:

1. they provide more pattern specification possibilities, and

*Research Assistant of the NFWO
†Department of Mathematics and Computer Science, University of Antwerp (UIA), Universiteitsplein 1, B-2610 Antwerp, Belgium, Email: {gemis,pareda,thyssens,vdbuss}@wins.uia.ac.be

SIGMOD /5/93/Washington, DC,USA
© 1993 ACM 0-89791-592-5/93/0005/0505...$1.50

2. they provide programming constructs such as loops and conditional operations.

Note that these macros are only abbreviation mechanisms.

To have a database manipulation system, we must be able to express database queries, updates and scheme manipulations in a uniform way, with the GOOD operations. This can be achieved by interpreting programs differently depending on their *execution-mode* [4]. There are four execution modes in GOOD: query, update, scheme manipulation and database restructuring. Any program can be run in any mode. The mode is chosen by the user, and describes whether the program works only temporary or changes only the database instance graph, or only the scheme graph, or both. In this way the user can express resp. queries, updates, scheme manipulations and database restructurings.

3 The user interface of GOOD

In the video we demonstrate two tools we have implemented to write GOOD programs and to display the result of database operations. The tools are written on DG AViiON computers and run under OSF/Motif.

The GOOD-programs contructed with these tools are translated into an adaptation of the relational algebra. This intermediate code can then be further translated into the DML and DDL of a specific database system (currently Informix ESQL/C.

The first tool, the *Program Builder tool* helps the user writing syntactically correct programs. It is a drawing program with graphical syntax-directed drawing capabilities, freehand drawing for example is not possible. The only way to "draw", is by invoking commands from a palette or menu-bar, so that the program can check immediately for syntactical correctness of the written program-step. Furthermore the user can either create new objects and properties or use existing objects and properties by copying them from a scheme graph. So the user can be sure that his program can be executed if he invokes the run-command.

The Program Builder tool allows the user to customize the layout of the scheme. Not only the position of the objects and relationships can be changed, the user can also split the graph into smaller ones or compose two unconnected subgraphs into one connected graph, if there are common objects. Exact definitions of composing and decomposing graphs can be found in [4]. Customized schemes can be stored and used in later programs.

The result of a program is examined with the second tool, the *Viewing tool*. In this tool, the user specifies what he wants to see, again by means of a pattern. This pattern is constructed in the same way as the patterns in a program step. The only difference is that the user does not need to add GOOD-operations to patterns in the Viewing tool. The matchings of the pattern are then displayed in tabular form. While a tabular format is a simple and natural way to represent structured data, other possibilities (e.g., graph-based or nested-tabular formats) are currently being investigated. The user can now select particular objects in the table and "pin them down" in subsequent patterns. This yields a novel and powerful technique for browsing [4] as an answer-driven process, naturally integrated in the viewing tool.

Both the Program Builder and the Viewing tool provide also additional features for pattern construction, besides copy and paste. Furthermore the selection of objects with the mouse is syntax-directed. The tools automatically extend the set of selected nodes and edges, so that this set is useful for the copy, cut and paste commands. In this way we avoid syntax errors in an early phase of the instruction-writing and speed up program construction.

More details on program building and querying with the Program Builder and Viewing tools

4 The Video

In the video session we demonstrate how one can contruct a program using the Program Builder. As an example we write a program that can be used to construct the scheme of a polygon database. Figure 1 shows the third step of this example. The user had selected the begin and end node of a new edge and clicked on the button with the edge label, so he/she can now type in the new edge label. The graph in the top window describes the scheme, while the graph in the bottom window specifies the GOOD-operation. In the video we show in detail how one writes a program, using the features offered by the Program Builder. We also show how one can modify the scheme layout with composition and decomposition. Then we show how one can construct queries with more complex patterns in the Viewing tool, and give an example of browsing. Figure 2 shows the Viewing tool with a pattern describing the query "Gives all polygons which have a vertex in point (6,4)", before the object identifiers of the polygons fulfilling the condition are displayed in the table. Finally we show an example of a GOOD-program specifying a more complex transformation that could be interpreted either as a query or as an update.

Acknowledgment: We like to thank Marc Gyssens and Dirk Van Gucht for their contribution in the design of the GOOD model. We also like to thank Marc Van der Linden for his contribution in the design of the GOOD to relational algebra interpreter. The video was made possible by financial support from Data General Belgium.

[1] J. Paredaens, J. Van den Bussche, D. Van Gucht, Gemis M., et al. An overview of GOOD. *ACM SIGMOD Record*, March 1992.

[2] M. Gyssens, J. Paredaens, and D. Van Gucht. A graph-oriented object database model. In *Proceedings of the Ninth ACM Symposium on Principles of Database Systems*, pages 417–424. ACM Press, 1990.

[3] M. Gyssens, J. Paredaens, and D. Van Gucht. A graph-oriented object database model for database end-user interfaces. In H. Garcia-Molina and H.V. Jagadish, editors, *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, number 19:2 in SIGMOD Record, pages 24–33. ACM Press, 1990.

[4] M. Andries, M. Gemis, J. Paredaens, I. Thyssens, and J. Van den Bussche. Concepts for graph-oriented object manipulation. In A. Pirotte, C. Delobel, and G. Gottlob, editors, *Advances in Database Technology—EDBT'92*, volume 580 of *Lecture Notes in Computer Science*, pages 21–38. Springer-Verlag, 1992.

[5] M. Gyssens, J. Paredaens, J. Van den Bussche, and D. Van Gucht. A graph-oriented object database model. Technical Report 92-35, University of Antwerp (UIA), 1992. Revised version of Technical Report no. 327, Computer Science Department, Indiana University, and of UIA Technical Report 91-27.

[6] J. Van den Bussche, D. Van Gucht, M. Andries, and M. Gyssens. On the completeness of object-creating query languages. In *Proceedings 33nd Symposium on Foundation of Computer Science*, pages 372–379. IEEE Computer Society Press, 1992.

[7] M. Andries and J. Paredaens. Macro's for the GOOD-transformation language. Technical Report 91-20, University of Antwerp (U.I.A.), 1991.

[8] M. Gemis, J. Paredaens, and I. Thyssens. A visual database management interface based on GOOD. In *Proceedings International Workshop on Interfaces to Database Systems*, 1993. To appear.

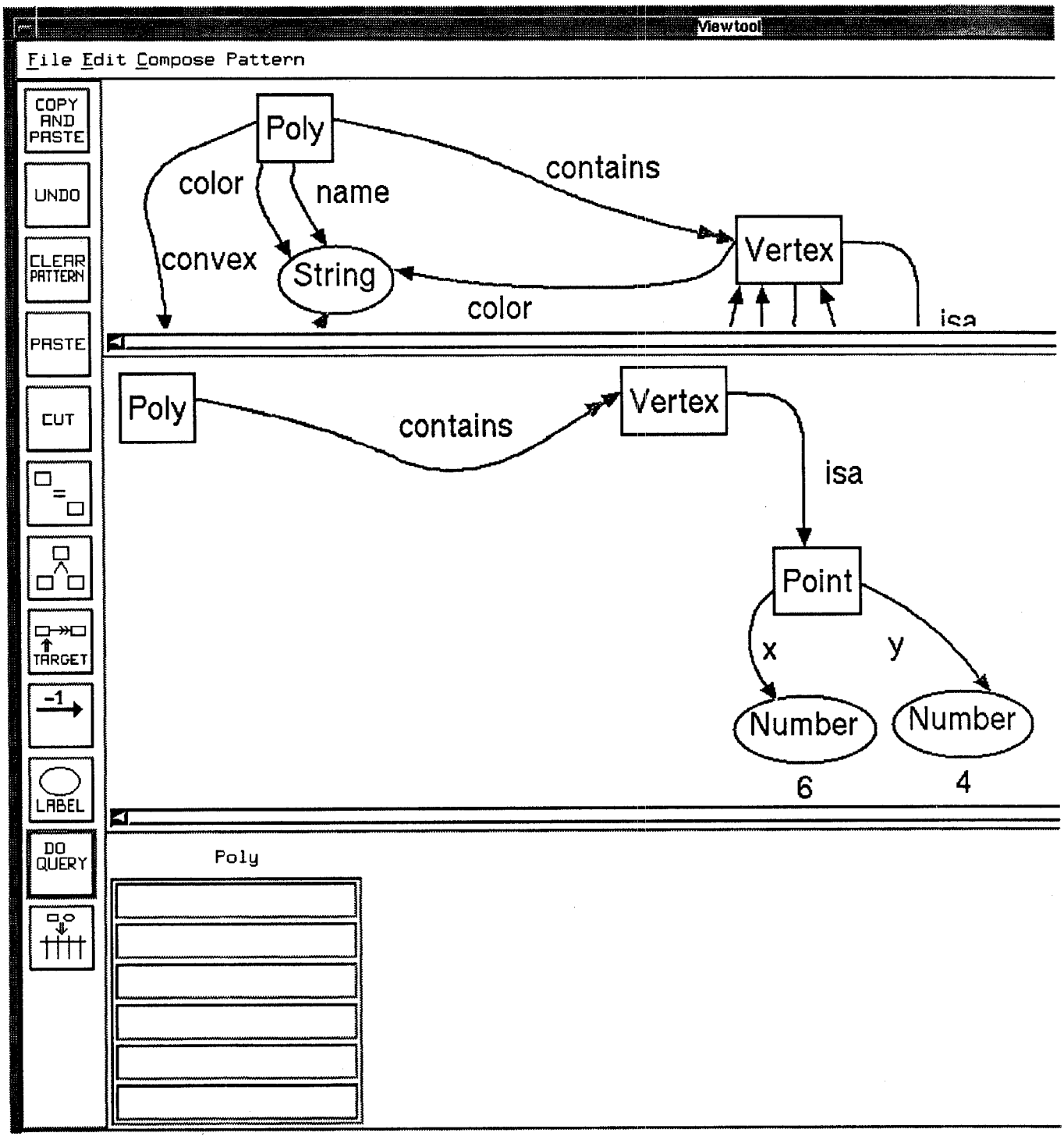

Figure 2: An example query in the Viewing Tool.

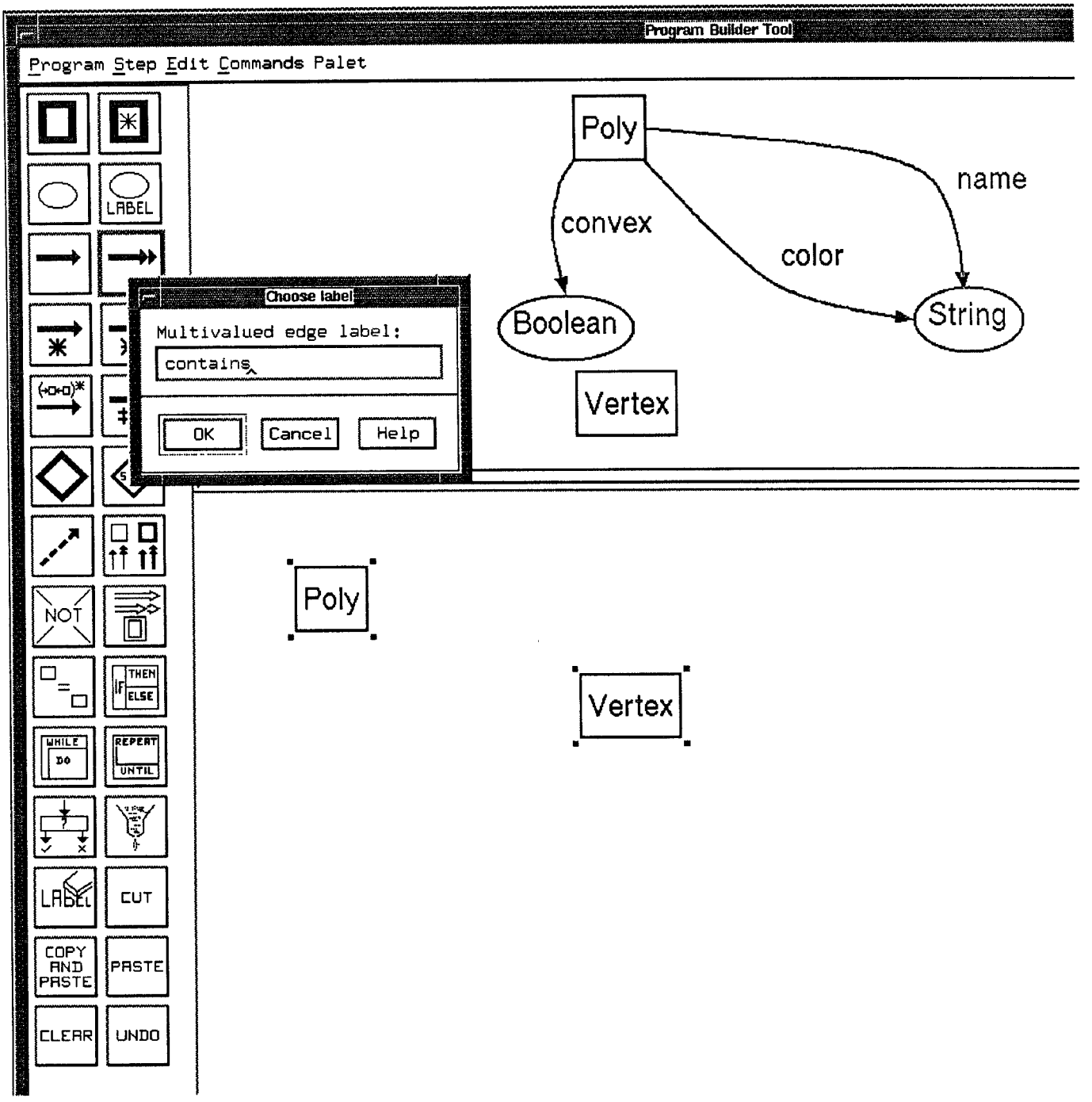

Figure 1: Construction of a GOOD-program step with the Program Builder.