Visualizing Queries and Querying Visualizations

Mariano P. Consens
consens@db.toronto.edu

Isabel F. Cruz
isabel@db.toronto.edu

Alberto O. Mendelzon
mendel@db.toronto.edu

Computer Systems Research Institute
University of Toronto
Toronto, Canada M5S 1A4

1 Introduction

Suppose you are flipping channels on a TV set and flip into the middle of a film that you have seen before. Most people can, within a few seconds, identify the film, recall the title and main actors, and predict what is going to happen next. Alan Kay [Kay91] gives this example to show the remarkable powers of storage and retrieval of visual information that human beings have. We may perhaps one day be able to duplicate some of this power in our computing machines, although the challenges are enormous; in the meantime, computer systems that manage, retrieve and manipulate information should be designed to play to the strengths of human information processing capabilities.

Presenting information visually whenever possible is one obvious way of doing this. Visual displays of quantitative information have of course been used for centuries in the physical and social sciences. Researchers in scientific visualization (see for example [MDB87]) are currently exploiting computer graphics technology to achieve dramatic improvements in the ability of scientists to understand the data with which they work. A less obvious idea is to provide data manipulation tools that are themselves visually oriented. The iconic user interfaces that are common in today's workstations are an example of this; visual query languages for databases are more ambitious tools for visual data manipulation.

In this paper, we describe the approach to visual display and manipulation of databases that we have been investigating at the University of Toronto for the past few years. We present an overview and retrospective of the G^+project [CMW87, CMW88, CM90b, CM90c] and the GraphLog query language by commenting a few screen shots showing different aspects of the work, and referring to the literature for the theory behind the prototype. We then point out some challenging research issues that show that we have just scratched the surface of what is sure to be a key aspect of future database systems.

2 The G^+/GraphLog Visual Query System

Retrieving information from databases involves an expression describing the relevant data (the *query*), some abstract description of the structures used to store the information (the *schema*), the actual data populating the database (the *input instance*), and the data that constitutes the answer to the query (the *output instance*). The final step in the querying process is the presentation of the output instance to the user.

Most of the work in visual query languages (see [BCCL91] for a recent survey) has overlooked instance visualization and instead has emphasized schema and query visualization. In the remainder of this section we will present several screen shots from a session with the G^+/GraphLog Visual Query System [CKM91]. We will discuss, through the examples, how we have used visualizations in this project: query visualization, input instance visualization, and several options for visualizing the output instance. Furthermore, from any of the visualizations presented to the user, she has an option to start browsing the structure being visualized in Hypermedia-like fashion.

The visualizations supported by the system are labelled graphs and *hygraphs*[1], all of which can be edited (including copy, cut and paste; panning and zooming; automatic layout; moving graphs to and from files; editing of node and edge labels; etc.).

The visual queries supported by the G^+/GraphLog Visual Query System are expressions of the GraphLog query language [Con89, CM90b]. GraphLog queries are graph patterns with nodes labelled by sequences of variables and constants, and whose edges are labelled by *path regular expressions* on predicates. The query

[1] Hygraphs [Con91] are a hybrid between Harel's higraphs [Har88] and directed hypergraphs, hence the name.

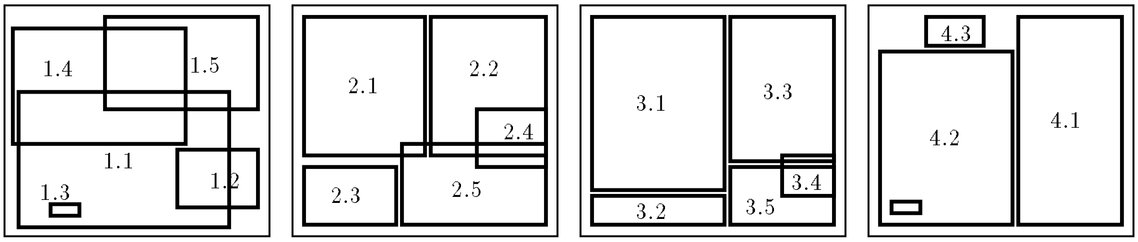

Figure 0: Map showing the numbers of the windows in the screen shots.

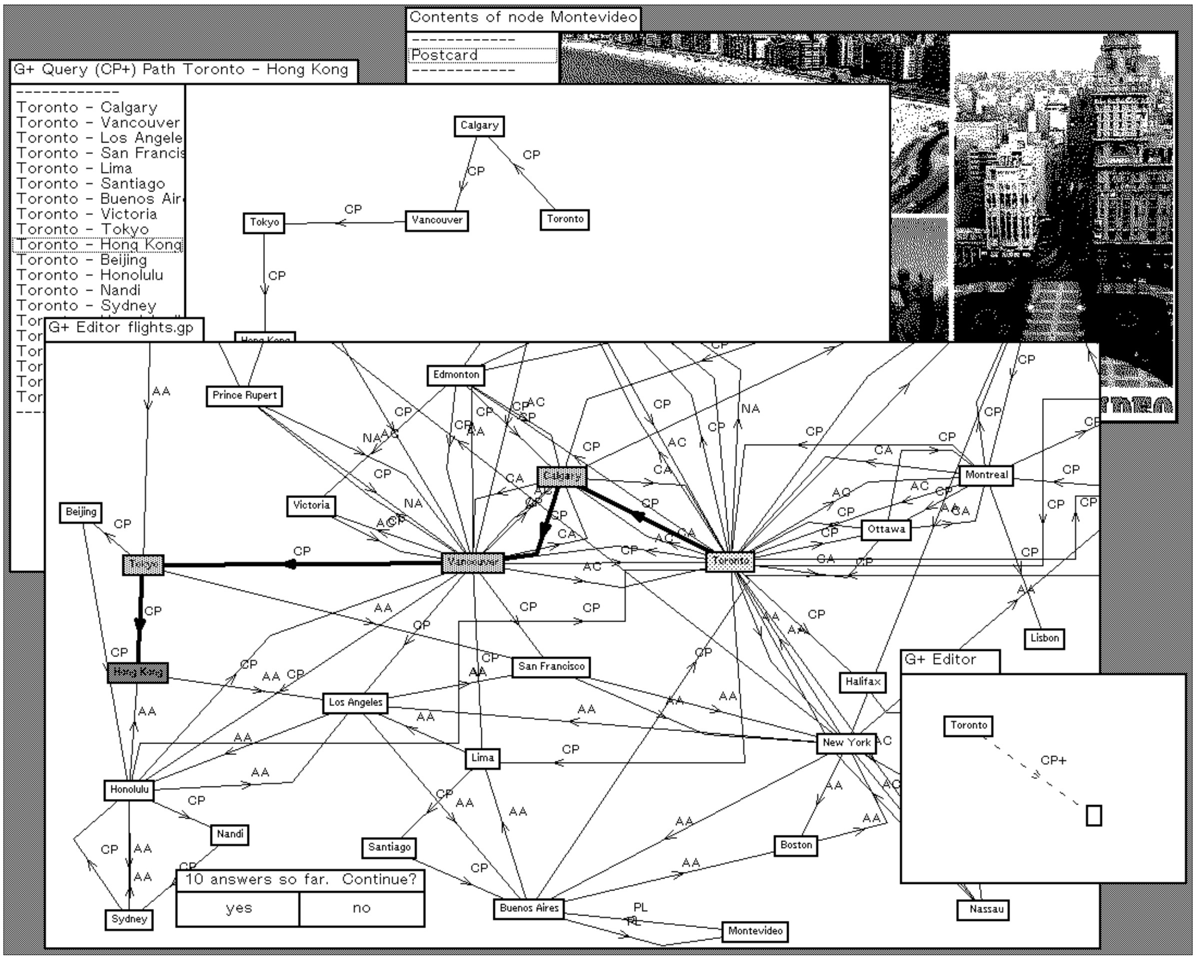

Figure 1: Showing flight connections in context and individually.

evaluation process consists of finding in the database all instances of the given pattern and for each such instance performing some action, such as defining a new arc in the database graph, or extracting from the database the instance of the pattern. GraphLog has higher expressive power than SQL; in particular, it can express, with no need for recursion, queries that involve computing transitive closures or similar graph traversal operations. The language is also capable of expressing first order aggregate queries as well as aggregation along path traversals (e.g., shortest path queries)[CM90c]. Precise theoretical characterizations of the expressive power of GraphLog and of its computational complexity can be found in the references cited above.

2.1 Visual Queries with Visual Answers

A portion of the visualization of a flights database instance appears in Window 1.1 of Figure 1 (please refer to Figure 0 for the numbering of screen windows). The flights database contains information about a

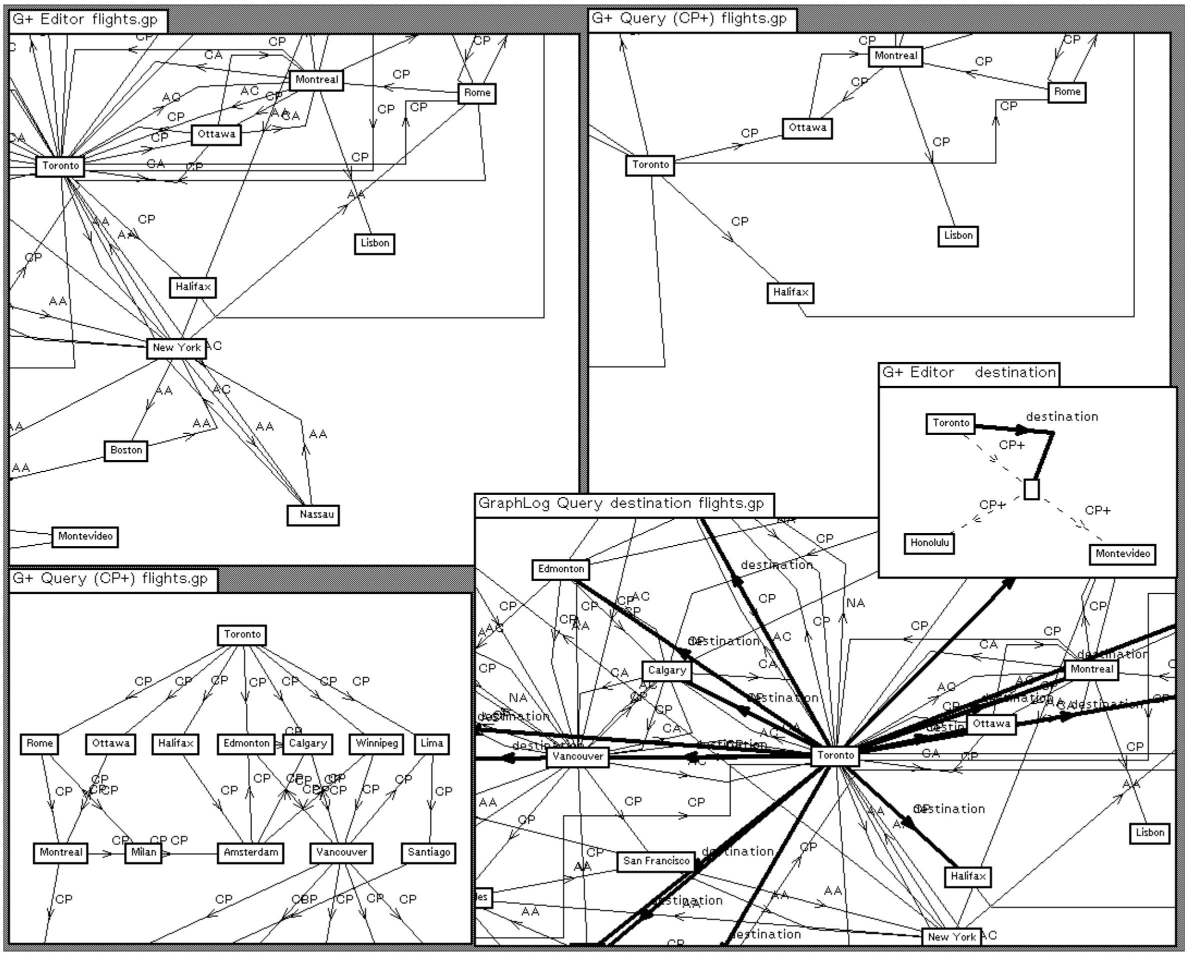

Figure 2: Changing the layout for some interesting flights and displaying new connections.

few airlines that connect several cities; its instance visualization has city names labelling nodes, and airline codes labelling edges. This graph overview can be used as a starting point for navigating and inspecting the contents of the database. For instance, a postcard for one of the cities is displayed in Window 1.5.

The pattern in Window 1.2 matches paths of CP (Canadian Pacific) flights originating in Toronto. Evaluation of the visual query in Window 1.2 is showing one possible trip with CP from Toronto to Hong Kong superimposed on the input instance shown in Window 1.1. This is the tenth answer being shown to the user, after being prompted for them one by one as shown in Window 1.3. Window 1.4 shows an alternative way of displaying the answer to the same query by collecting in a separate window all paths that answer the query. In this case, the same trip from Toronto to Hong Kong is being visualized in isolation, and we can see a list with 20 other possible trips out of Toronto.

Windows 2.1 and 2.2 in Figure 2 provide a comparison between the same section of the flights database before and after it has been filtered with the query in Window 1.1. Filtering retains only those cities and flights that appear along CP flights paths out of Toronto, selectively reducing the amount of information displayed. Window 2.3 illustrates the usefulness of layout algorithms by topologically sorting the filtered database of Window 2.2, hence providing visual information about cities that can be reached in a given number of stops from Toronto (in this context, layout can be seen as a two-dimensional analog of sorting the answer to a relational query).

The query visualized in Window 2.4 has a pattern with the bold edge labeled destination defining a binary relation between Toronto and those cities reachable from Toronto from where both Montevideo and Honolulu are reachable (always using CP flights). The answer to the above query against the database in Window 1.1 appears (superimposed to the original database) in another window (Window 2.5).

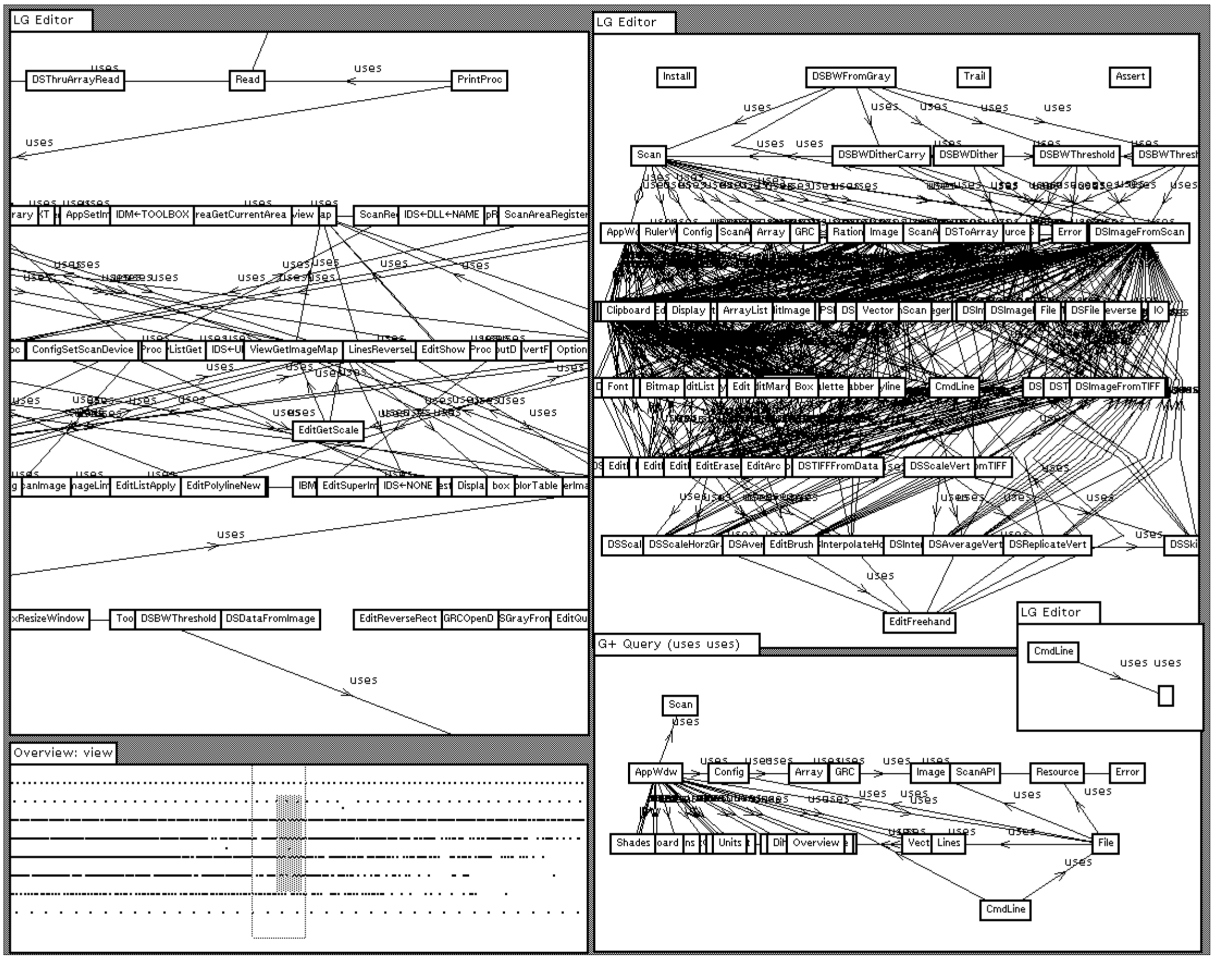

Figure 3: Displaying overviews of large software graphs and filtering relevant portions.

2.2 Dealing with Large Data Visualizations

The flights database that we have been using is just a toy. The G$^+$/GraphLog Visual Query system has been used to visualize information extracted from relational databases, Hypertext systems [CM89], and large software repositories, involving graphs with over 10,000 nodes and edges.

Figure 3 shows a database containing information about the packages involved in a software system (see [CMR91] for applications of GraphLog to software engineering). Package names appear in the nodes, while the edges describe the usage relationship between packages. There are approximately one thousand edges and nodes in this example. In order to deal with these information rich displays, the system supports the usual technique of providing overviews at different levels of detail. This is illustrated by the overview in Window 3.2, where nodes are visualized as dots and no edges are displayed. Window 3.1 shows in detail the rectangle that appears shaded in the overview window. However, the use of filtering (from Window 3.3 to Window 3.5 by using the query in Window 3.4) is the most powerful tool contributed by the system to the management of large data visualizations.

2.3 Managing Complexity

Figure 4 presents the hygraph version of the flight database in Window 4.2 (together with a query in Window 4.3) as well as another software example in Window 4.1. The software hygraph has nodes labelled by module and function names, while the relationship labelling the edges corresponds to calls among functions. The *blobs* (boxes) labelled contains represent containment between modules or containment of functions by modules. Similarly, the flights database has been extended to include relationships like country, province, city that are visualized as blobs. Note that some of the blobs are shown without their contents being visible (e.g., the module marqwher). Interactively hiding and showing blob contents illustrates one of the ways in which

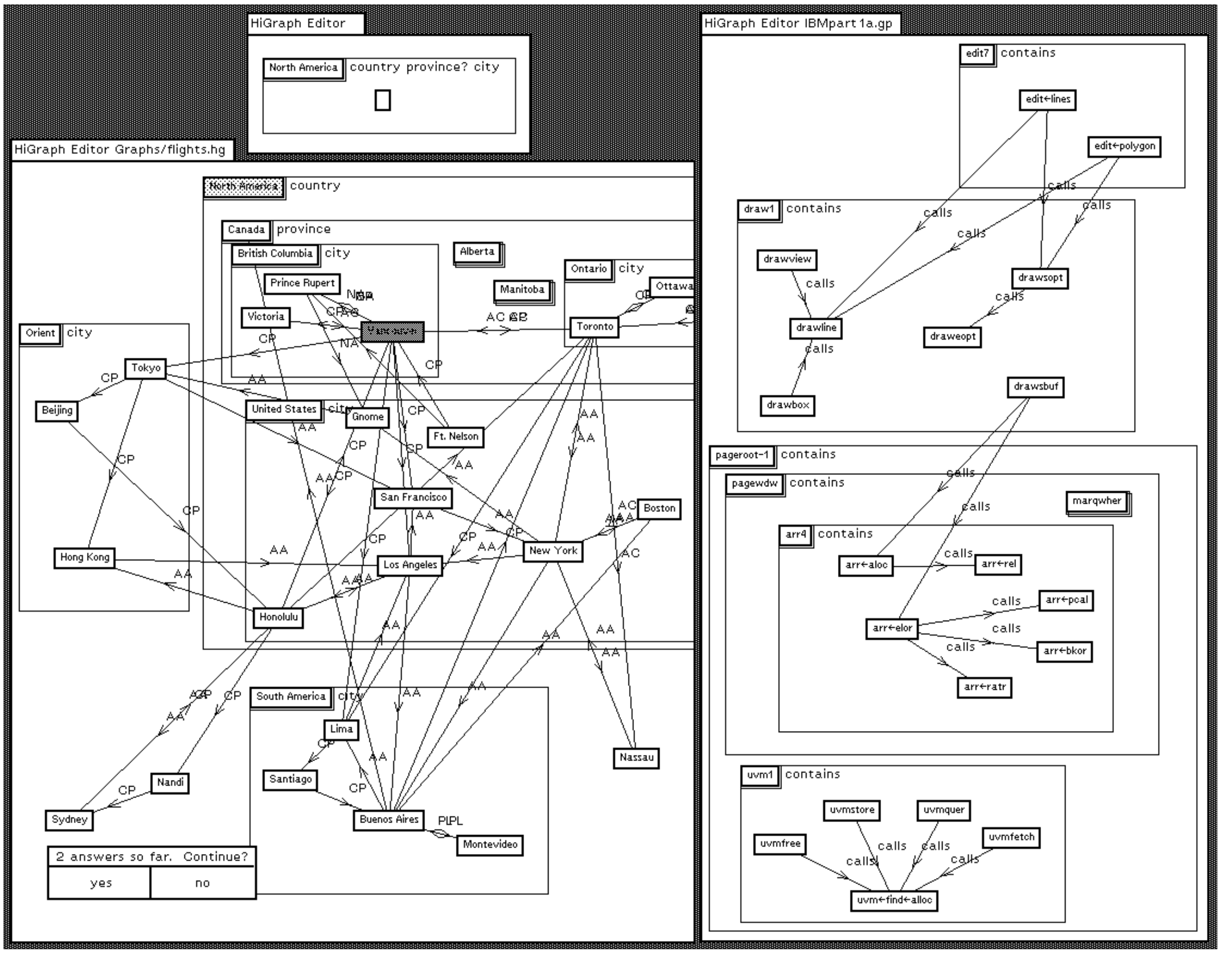

Figure 4: Hygraphs for the flight and software engineering examples.

hygraphs allow for varying levels of abstraction in the display of hierarchical data.

There is an interesting contrast between the examples: while flights have a natural geographic interpretation that makes their visualization on a map obvious, the example of Window 4.1 attempts to provide a visualization of the intangible and invisible: software.

In retrospect, the strengths of the G^+/GraphLog Visual Query System have been the visual nature of the query process, the variety of visual answer modes supported, and having a query language with formal semantics and known expressive power.

3 Research Issues

3.1 User-Defined Visualizations

There are two natural ways to generalize the G^+ work described in the previous section. First, the data model of GraphLog is relational; we would like to extend it to richer formalisms such as object-oriented models. Second, the visualization is fixed by the system and consists of graphs and hygraphs; we would like to have a multitude of different visualizations available to allow the application designer or end user to tailor the visualization to the data and to the application, or to switch at will between different ways of displaying the same data.

To address these goals, the authors are currently investigating declarative and visual languages for object-oriented databases. One particular approach [Cru91], whose underlying principle is *display and query arbitrary data with arbitrary pictures,* is sketched in the next two subsections.

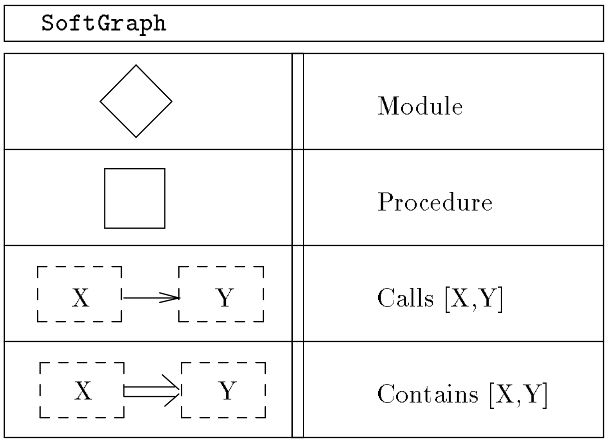

Figure 5: Template for displaying a software engineering database.

Specifying visualizations of object-oriented databases

A *visualization* is a set of conventions for obtaining pictures from data. We want to be able to display more advanced models of data than the relational model, and give freedom to the user to define her favorite visualization. These visualizations are as little hard-wired as possible. For example *graph* is not a display primitive. Figure 5 illustrates how the user can create a graph visualization for an object-oriented software engineering database by means of a visual program. The name of the user-defined visualization is `SoftGraph`.

A visual program in our language is a set of visual rules or *rows*. These rows are to be read from right to left, in a way that is similar to other deductive query languages. The first two rows specify that instances of the class Module (resp. Procedure) are to be displayed as diamonds (resp. squares). The other rows specify that instances of the class Call (resp. Contains) are to be displayed as simple (resp. double) arrows. The semantics of the language are defined by mapping queries into rules in a declarative object-oriented language similar to the ones surveyed in [Cru90].

Our visual language is object-oriented. For instance, a dashed rectangle, as in Figure 5, refers to the display of an object regardless of its class. The specific display depends on the class to which the object belongs, and can be inherited from an object's ancestral class. Using this language the user can specify other visualizations so that the data can be displayed as a graph, a hygraph, a bar chart, a temporal chart, etc.

Querying and Transforming Visualizations

In the previous subsection we expressed a query as a function mapping databases to visualizations; more generally, we can think of the visualizations themselves as part of the database (as in [AEM86],) and use our visual language to query the visualizations and produce different visualizations as result. Figure 6 (i) shows a template in our visual language that transforms a graph to a temporal chart.

A temporal chart consists of two or more parallel axes and a set of lines between these axes. Temporal charts are a successful way of visually displaying flight schedules, task accomplishment, interacting concurrent processes, and so on [Tuf90]. In the following example, concerning flight schedules, each axis represents a place (of departure, say a, or arrival, say b), and has a time scale associated with it. A line is drawn between a point t on the axis associated with a and a point t' on the axis associated with b if there is a trip starting at time t from place a, and arriving at time t' to place b.

The first row of the template specifies that a node of the graph should be placed on the axis that represents the place of departure or of arrival. The second row specifies that nodes on an axis are placed according to a scale. The third row specifies that the origin of the two axes is to be placed at the same horizontal coordinate.

To choose a flight from Rome to Toronto based on the elapsed time from departure to arrival, the data display of Figure 6 (iii) conveys more information at a glance than the original graph display of the data of Figure 6 (ii).

There is a rich theory of the *expressive power* [Cha88] of database languages, developed initially for

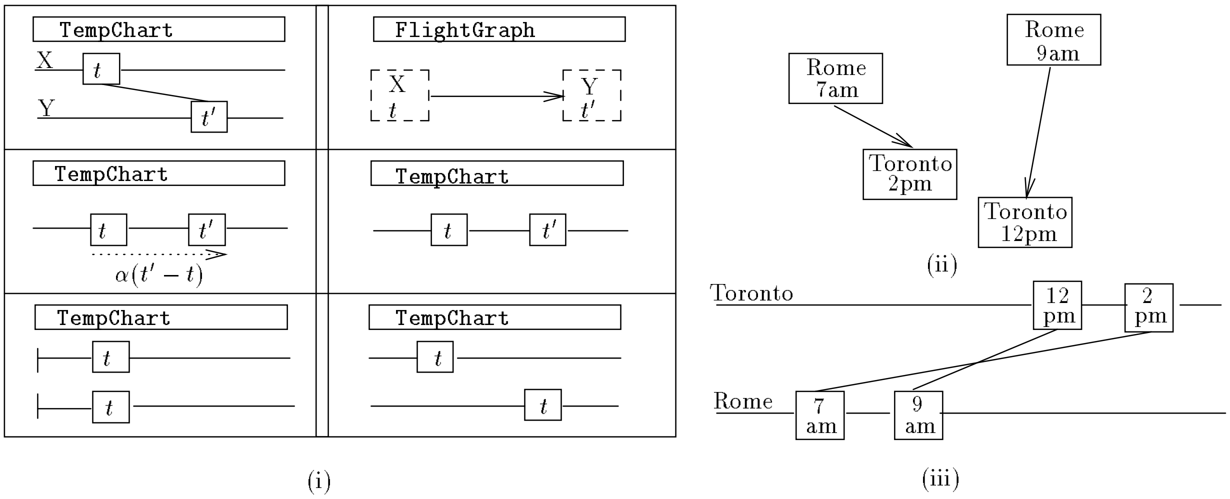

Figure 6: Graph to temporal chart transformation.

relational languages and extended more recently to complex object models [Cru90]. What is the expressive power of a language for specifying and manipulating visualizations? How do we compare, say, a language that can specify only graphs with one that allows only temporal charts?

3.2 Managing Visual Complexity

Our experience with the G$^+$/GraphLog Visual Query System quickly showed us that naïve approaches to graph visualization do not scale up. Graphs with more than a hundred or so nodes and edges are difficult to display on today's workstation screens in a way that is meaningful.

All the techniques discussed in Section 2.3 work by simplifying the visualizations. One alternative perspective on the management of visual complexity is provided by Tufte [Tuf90]. He downplays the importance of visual simplicity, emphasizing instead the design of "information rich" displays:

> So much for the conventional, facile, and false equation: simpleness of data and design = clarity of reading. Simpleness is another aesthetic preference, not an information display strategy, not a guide to clarity. What we seek instead is a rich texture of data, a comparative context, an understanding of complexity revealed with an economy of means.

Some authors (notably [CRM91]) have proposed rich visualizations of complex data, such as the *cone tree* and the *perspective wall*, without an explicit underlying data model and query language. To make such techniques part of the tool kit of application designers, we need to integrate them into a data model and to be able to specify transformations between these and other visualizations.

3.3 Good and Bad Visualizations

Mackinlay [Mac86] points out that a visualization, just like a logical theory, contains both explicit and implicit information. A mismatch between the facts implicit in the data and those implicit in the visualization may lead to two kinds of problems. If not all facts implied by the data are implied by the visualization, then the visualization is not *complete*. Of course, incomplete visualizations are not necessarily incorrect, as incompleteness is one way of dealing with visual complexity. The converse is when the visualization implies facts that are not valid consequences of the data, a "lossy join"-like phenomenon. The canonical example given by Mackinlay is the use of nested boxes to represent non-transitive relationships. In this case, the visualization is not *sound*.

More work needs to be done in characterizing the information content of a visualization. Beyond soundness and completeness is the issue of *effectiveness*: equally expressive visualizations may differ in their ability to convey information effectively to human users. Existing knowledge on human perception, as well as empirical testing, must be taken into account in evaluating the effectiveness of a visualization for a particular task.

Acknowledgements

We would like to thank Frank Eigler, Christine Knight, Carlos Mendioroz, Arthur Ryman, Peter Wood and Annie Yeung for their many contributions to the G^+/GraphLog project. Yannis Ioannidis made many helpful comments on an earlier draft of this paper.

This work has been supported by the Information Technology Research Centre of Ontario, the Natural Sciences and Engineering Research Council of Canada, and the Centre for Advanced Studies, IBM Canada Laboratory, and the IBM Shared University Research Program. The first author was also supported by the PEDECIBA (United Nations Program for the Development of Basic Sciences in Uruguay) and by an IBM Canada Research Fellowship. The second author was supported in part by an INVOTAN grant and a Government of Canada Award.

References

[AEM86] T. Lougenia Anderson, Earl F. Ecklund,Jr., and David Maier. PROTEUS: objectifying the DBMS user interface. In *Intl. Workshop on Object-Oriented Database Systems*, 1986.

[BCCL91] C. Batini, T. Catarci, M. F. Costabile, and S. Levialdi. Visual query systems. Report 04.91, Universita degli Studi di Roma La Sapienza, March 1991.

[CRM91] Stuart K. Card, George G. Robertson, Jock D. Mackinlay. The information visualizer, an information workspace. In *Proceedings of CHI'91*, pages 181-188, 1991.

[Cha88] Ashok K. Chandra. Theory of database queries. In *ACM Symposium on Principles of Database Systems*, pages 1–9, 1988.

[CKM91] Mariano P. Consens, Christine N. Knight, and Alberto O. Mendelzon. The architecture of the G^+/GraphLog visual query system. Technical Report TR 74.054, IBM Canada Laboratory, 1991.

[CM89] Mariano P. Consens and Alberto O. Mendelzon. Expressing structural hypertext queries in GraphLog. In *Proceedings of the Second ACM Hypertext Conference*, pages 269–292, 1989.

[CM90b] Mariano P. Consens and Alberto O. Mendelzon. GraphLog: a visual formalism for real life recursion. In *Proceedings of the Ninth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 404–416, 1990.

[CM90c] Mariano P. Consens and Alberto O. Mendelzon. Low complexity aggregation in GraphLog and Datalog. In *Proceedings of the Third International Conference on Database Theory, Lecture Notes in Computer Science Nr. 470*, pages 379–394. Springer-Verlag, 1990.

[CMR91] Mariano P. Consens, Alberto O. Mendelzon, and Arthur G. Ryman. Visualizing and querying software structures. Technical Report TR 74.053, IBM Canada Laboratory, September 1991.

[Con89] Mariano P. Consens. Graphlog: "real life" recursive queries using graphs. Master's thesis, Department of Computer Science, University of Toronto, 1989.

[Con91] Mariano P. Consens. Towards a theory of visual manipulations of database visualizations. Technical Report, Department of Computer Science, University of Toronto (forthcoming), 1991.

[Cru90] Isabel F. Cruz. Declarative query languages for object-oriented databases. In F. H. Lochovsky, editor, *Office and Data Base Systems Research '89*, pages 92–130. Technical Report CSRI-238, June 1990.

[Cru91] Isabel F. Cruz. Defining and querying visualizations of data. Technical Report, Department of Computer Science, University of Toronto (forthcoming), 1991.

[CMW87] Isabel F. Cruz, Alberto O. Mendelzon and Peter T. Wood. A graphical query language supporting recursion. In *Proc. ACM-SIGMOD 1987 Annual Conference on Management of Data*, pages 323–330, 1987.

[CMW88] Isabel F. Cruz, Alberto O. Mendelzon and Peter T. Wood. G^+: recursive queries without recursion. In *Proc. 2nd International Conference on Expert Database Systems*, pages 355–368, 1988.

[Har88] David Harel. On visual formalisms. *Communications of the ACM*, 31(5):514–530, 1988.

[Kay91] Alan Kay. Interviewed in *BYTE* 16:11, 1991, p.43.

[Mac86] Jock D. Mackinlay. Automatic Design of Graphical Presentations. Technical Report STAN-NCS-86-1138, Department of Computer Science, Stanford University, 1986.

[MDB87] Bruce H. McCormick, Thomas A. DeFanti, and Maxine D. Brown (editors). Visualization in scientific computing. In *SIGGRAPH Computer Graphics* 21:6, November 1987, entire issue.

[Tuf90] Edward R. Tufte. *Envisioning Information.* Graphics Press., Cheshire, Conn., 1990.