Hy+: A Hygraph-based Query and Visualization System

Mariano P. Consens
consens@db.toronto.edu

Alberto O. Mendelzon
mendel@db.toronto.edu

Computer Systems Research Institute
University of Toronto
Toronto, Canada M5S 1A1

1 Introduction

This paper provides a brief overview of Hy+, a hygraph-based query and visualization system, which is presented in the accompanying video.

Hy+ provides a user interface with extensive support for visualizing structural (or relational) data as *hygraphs* (of which labelled graphs are a special case). Hygraphs, defined in [Con92], are a hybrid between Harel's higraphs [Har88] and directed hypergraphs. Hygraphs constitute a convenient abstraction that generalizes several diagrammatic notations.

The Hy+ system supports visualizations of the actual *database instances*, and not just diagrammatic representations of the database schema. Given the large volume of data that the system must present to the user, it is fundamental to provide her with two fundamental capabilities.

The first one is the ability to *define* new relationships by using queries. This is pretty much the traditional way of using database queries: the newly defined relationship either gives a direct answer to a user question, or it provides a new view on the existing data. The derived data can later be presented visually by the system.

The second capability is an innovative way of using queries to decide what data to *show*. Using this capability the user can selectively restrict the amount of information to be displayed. This *filtering* of relevant information is fundamental if one is to have any hope of conveying manageable volumes of visual information to the user. Selective data visualization can be used to locate relevant data and to restrict visualization to interesting portions of it (i.e., deciding *what* data to present), and can also be used to control the level of detail at which the data is presented (i.e., choosing *how* to see the data).

To describe queries, Hy+ relies on a visual pattern-based notation. The patterns are expressions of the GraphLog query language [Con89, CM90b]. Overall, the system supports query visualization (i.e., presenting the description of the query using a visual notation), the (optional) visualization of data that constitutes the input to the query, and the visual presentation of the result.

An earlier version of this paper appears in Proceedings of the 1993 ACM SIGMOD Conference, pages 511–516.

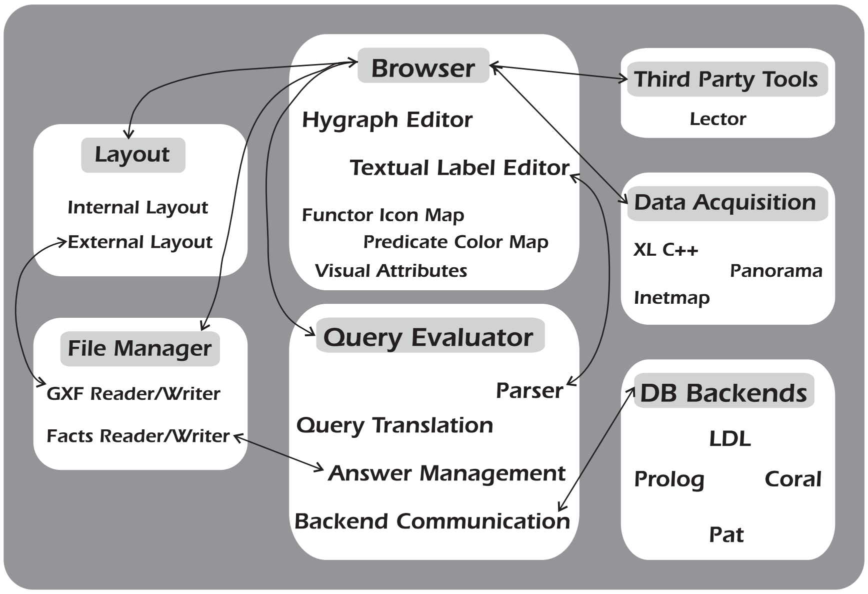

Figure 1: Hy$^+$ Architecture Overview.

Hy$^+$ has evolved from the G$^+$ Visual Query System presented in [CM90a]. The new system is implemented as a front-end, written in Smalltalk, that communicates with other programs to carry on its tasks (including multiple database backends for the actual evaluation of the queries).

An overview of the Hy$^+$system architecture is given in the diagram in Figure 1. The remaining sections of this paper are organized according to the main components in that diagram.

2 Data Acquisition

In the first section of the video we demonstrate Hy$^+$ by describing how a software engineer can use the system to visualize and query a large C++ code library (the NIH public domain library, which defines standard data structures for C++). The IBM XL C++ compiler was used to extract 13,000 facts from the 35,000 lines of code in the library.

The Hy$^+$ system relies on other programs (which are part of the Data Acquisition module) to supply the raw data to be visually manipulated within the system. The File Manager module can directly import files containing logical facts (like the ones produced by the XL C++ compiler). These files can also be obtained from relational and deductive databases. In addition, the system supports the GXF file format [Eig93], developed internally, which describes not just the logical facts, but also all the positioning and visualization-related information that completely defines the appearance of the data on the screen. Using GXF, a data acquisition tool can provide Hy$^+$ with a specific visualization as a starting point for the querying process supported by the system.

The final section of the video presentation provides a glimpse of other applications of Hy$^+$ to visualize and query data from different domains: formal software design documentation and code overlay structure ([CMR92]), debugging distributed and parallel programs ([CHM93]), and network management ([CH93]).

3 Hygraph Browsers

Hy$^+$ provides browsers that let users interact with the hygraph-based visualizations that the system manipulates. A hygraph extends the notion of a graph by incorporating *blobs* in addition to edges. A blob relates a containing node with a set of contained nodes, instead of relating a node to another node as edges do. Blobs are diagrammatically represented by a closed curve that is associated with the container node and that encloses the contained nodes.

Formally, a *hygraph* H is a septuple

$$(N, L_N, \nu, L_E, E, L_B, B)$$

where: N is a finite set of *nodes*; L_N is a set of *node labels*; ν, the *node labelling function*, is a function from N to L_N that associates with each node in N a label from L_N; L_E is a set of *edge labels*; $E \subseteq N \times N \times L_E$ is a finite set of *labelled edges*; L_B is a set of *blob labels*; and $B \subseteq N \times 2^N \times L_B$ is a finite set of *labelled blobs*. A restriction can be placed in the labelled blobs relationship B to ensure that there is only one tuple (n, N, l) in B with the same values for the *container node* n and the blob label l (i.e., the container node and blob label values functionally determine the value of the set of *contained nodes* N).

A major difference between hygraphs [Con92] and Harel's higraphs [Har88] is that the former are merely syntactic objects that do not provide any semantics to the relationships among nodes, edges and blobs (i.e., they are as uninterpreted as graphs are), while the latter imposes a fixed interpretation for the objects in a higraph (blobs represent sets, and the containment relationship is interpreted as set membership). The attractiveness of hygraphs comes from their flexibility to assign any semantics to the relationship represented by the blobs (the meaning will be based on some interpretation of the blob labels). Also, there can be more than one blob associated with a given node (in the same way that there can be multiple edges from a node).

Within Hy$^+$, all blobs associated with one container node are represented by rectangles contained within a rectangular region that has the container node in the topmost left corner. Blob labels are drawn in the interior of the topmost left corner of the rectangle representing the blob. In addition, the system enforces a strict hierarchy of nested blob rectangles by creating, if necessary, multiple occurrences of a node with the same label (i.e., it ensures that the blob-induced relationship is a forest). Used in this way, hygraphs constitute a flexible mechanism for clustering information and they support varying levels of abstraction in the display of both hierarchical and non-hierarchical data. Additional control over the level of detail displayed is achieved by interactively hiding and showing blob contents.

Hygraphs are a convenient formal abstraction for several styles of diagrammatic data presentations, not just graphs and extensions. By ignoring edges, and making use of multiple blobs associated with the same node as well as recursive node containment, one can easily model traditional (nested) form-based presentations. In the context of hypermedia presentations, the edges in the hygraph can be interpreted as *links* while the blobs can be

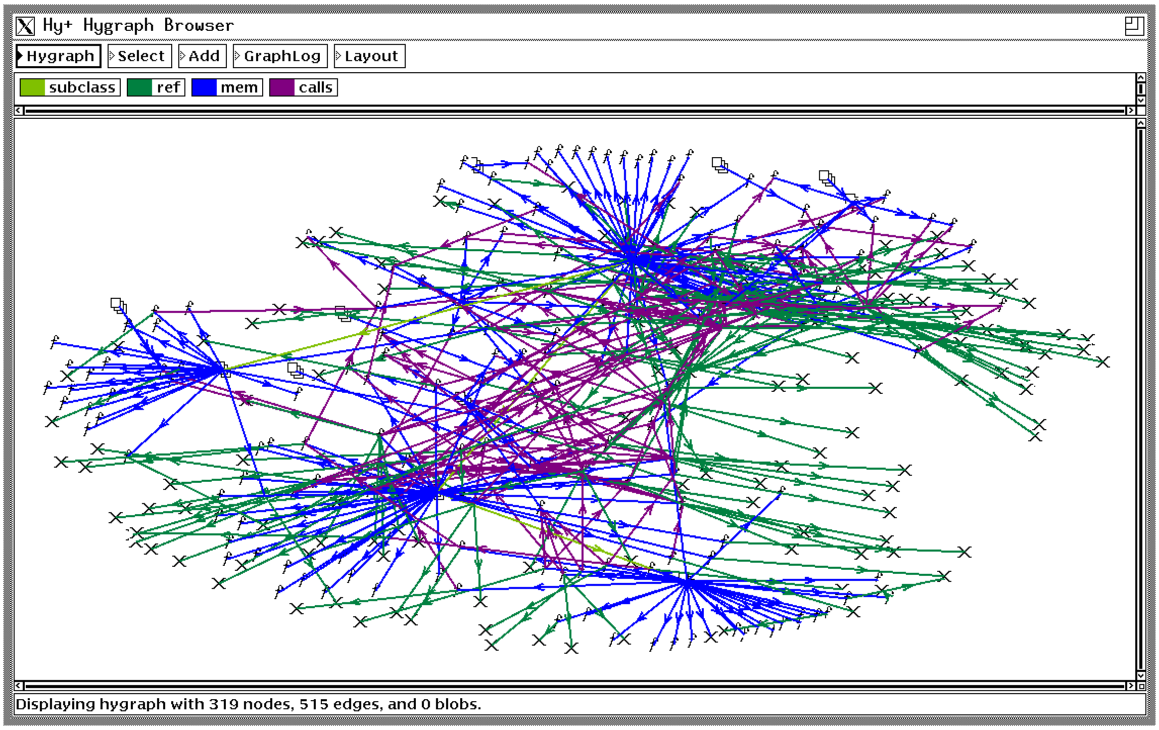

Figure 2: An uninformative visualization.

regarded as *fat links*. This view of hygraphs, similar to Tompa's directed hypergraphs [Tom89], makes them a suitable abstraction of the navigational choices presented to the users that browse hypermedia-like information structures.

Database instances in several application domains, regardless of the underlying database model, can be naturally visualized as hygraphs. Object-oriented databases are especially suited to graph-like representations, and so are hypertext databases [CM89]. But even plain relational databases can be easily represented by hygraphs having an edge (resp. a blob) labeled $r(c_1, \ldots, c_k)$ from a node labeled $(a_1, \ldots, a_i)$ to a node (resp. containing a node) labeled $(b_1, \ldots, b_j)$ corresponding to each tuple $(a_1, \ldots, a_i, b_1, \ldots, b_j, c_1, \ldots, c_k)$ of each relation r in the database.

Hy$^+$ browsers have extensive facilities for interactively editing hygraphs, including copy, cut and paste; panning and zooming; and textual editing of node and edge labels. Icons are automatically selected for nodes according to the functor of the node label (i.e., the type of data object represented by the node). Similarly, the colours for edges and blobs are selected based on the predicate in the label (i.e., the relationship represented by the edge or blob).

Figure 2 has one Hy$^+$ browser displaying a small portion of the NIH database mentioned in the previous section. Four relations are being displayed (their corresponding colours are displayed on the top pane, where they can be changed). The edges indicate, depending on the predicate labelling them, whether a class is a subclass of another one, or a function is a member of a class, or a function references a variable, or a function calls another function.

It is clear that just displaying a graph (despite the use of a *spring* layout algorithm to position the nodes) does not magically produce a useful visualization. In the next section the reader can get a glimpse of Hy$^+$ approach to generate informative database visualizations.

4 Hy$^+$ Query Evaluation

The visual queries supported by the Hy$^+$ system are expressions of the GraphLog query language [Con89, CM90b], suitably extended to hygraphs. GraphLog queries are hygraphs whose nodes are labeled by sequences of terms, and whose edges and blobs are labeled by *path regular expressions* on relations. The query evaluation process consists of finding in the database all instances of the query hygraph and for each such instance performing some action, such as defining new arcs in the database hygraph, or extracting from the database the instances of the query hygraph. 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 (an example involving maximizing is given later in the paper) 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.

Formally, GraphLog *query hygraphs* are hygraphs with no isolated nodes having the following properties. The nodes are labeled by sequences of terms. Each edge and blob is labeled by a literal (either an atom or a negated atom) or by a *closure literal*, which is simply a literal s followed by the positive closure operator, denoted s^+. Closure literals may only appear between nodes labeled by sequences of the same length. Queries have one or more *distinguished* edges and blobs (drawn thicker), which can only be labeled by positive non-closure literals. A *graphical query* is a finite set of query hygraphs.

The semantics of graphical queries are given by a translation to stratified linear Datalog (extended in a straightforward way if function symbols are present). Each query hygraph H in a graphical query corresponds to a rule r for each distinguished edge and blob, with the label of the distinguished edge or blob in the head, and as many literals in the body as there are non-distinguished edges and blobs in G. An edge or blob of the query graph labeled with a closure literal s^+ introduces a predicate defined by additional rules expressing the transitive closure of the predicate in s. The body of r contains the predicates introduced by the closure literals and the remaining edge and blob labels of G.

We allow as expressions of the GraphLog query language only those graphical queries whose distinguished edges and blobs define non-recursive predicates. Note that, although we disallow explicit recursion, recursion is nevertheless implicit in the use of closure literals.

The language can be made considerably more concise by generalizing literals and closure literals to arbitrary regular expressions. Each operator introduced is definable in terms of the basic language and is added only for convenience. In addition to the usual operators for positive and Kleene closure, alternation, and concatenation, two new ones are defined: inversion reverses the direction of the edge or blob labeled by the regular expression, and negation negates the predicate defined by its argument.

The Hy$^+$ system interprets GraphLog patterns in two fundamentally different ways:

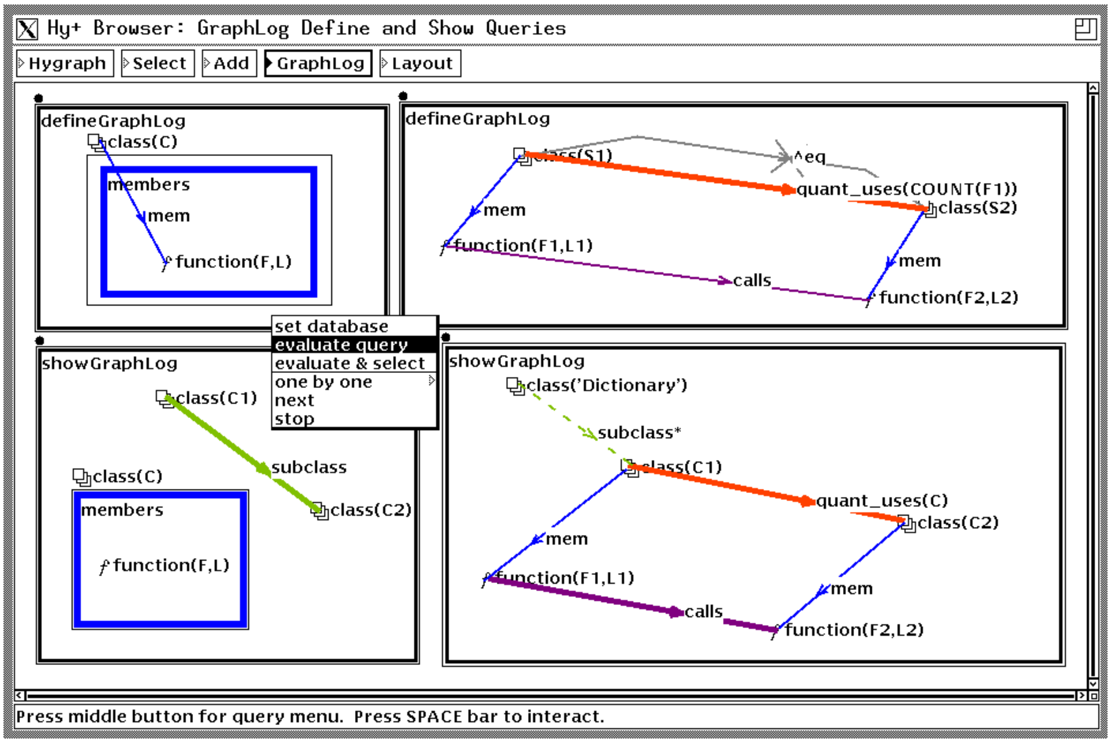

Figure 3: A Define and Show Query.

Define is the classical database interpretation of the pattern as a query that defines relations described by the distinguished edges or blobs.

Show (formally described in [Con92]) is at the core of Hy^+ capabilities for manipulating database visualizations, as opposed to just databases. Informally, when executing a query in show mode the pattern does not define any new relation, but it filters out information from the input visualization to produce an output visualization that contains the union of all the sub-hygraphs in the input that match the query pattern.

Figure 3 contains one Hy^+ query. There are four top level blobs in the query. Two of them, labelled defineGraphLog, enclose expressions which are interpreted as *define* queries. The one on the left defines (making use of GraphLog aggregation capabilities) a derived predicate that has the count of the number of calls from functions in one class to functions in another (therefore providing a measure of *coupling* among classes). The defineGraphLog blob on the right defines a blob that clusters with each class all its members functions. This simple query shows how easy is to change the visualization of a relationship from edges to blobs, and viceversa.

The showGraphLog blob in the right lower corner specifies that all member blobs and all subclass edges should be displayed. The one in the left lower corner is quite more specific about which quant_uses and calls edges should be displayed (only quant_uses edges outgoing from subclasses of class('Dictionary') and the associated calls). In general, a showGraphLog blob contains a hygraph pattern in which certain objects are distinguished.

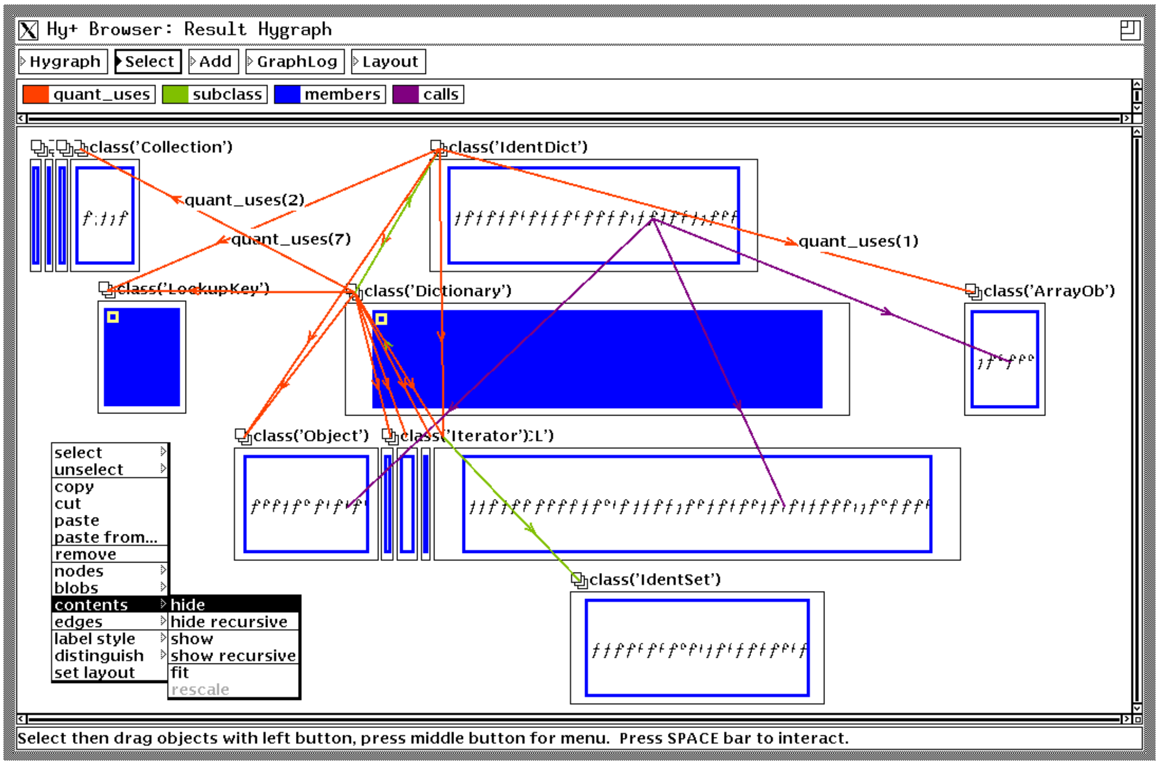

Figure 4: A Layout of the Answer to the Query.

For each such blob in a query, Hy$^+$ finds all instances of the pattern in the database and displays the corresponding distinguished objects in the visualization returned as the answer.

Hy$^+$ executes queries by translating the patterns into backend programs the that are evaluated against the current database hygraph. This translation, as well as the communication with the backend and the processing of the answers, are carried by the Query Evaluator component.

Currently, there are three back-end query processors which are part of the DB Backends component: LDL [NT89], Coral [Ram92], and a previously developed GraphLog interpreter implemented in Prolog [Fuk91]. The reader is referred to [CMV93] for a description of the use of deductive database technology in the Hy$^+$ system.

5 Hygraph Layout

The result of the query described in the previous section is displayed in Figure 4. The positioning of the nodes and blobs in the figure is the result of running a hierarchical layout algorithm. This algorithm is part of a suite of external layout programs [Noi93] that can be invoked from the External Layout component of the system. Note that the contents of two blobs have been interactively hidden to further reduce clutter. When hiding blob contents, the incoming and outgoing edges are also hidden (but there are options to present summaries of the information carried by the hidden edges).

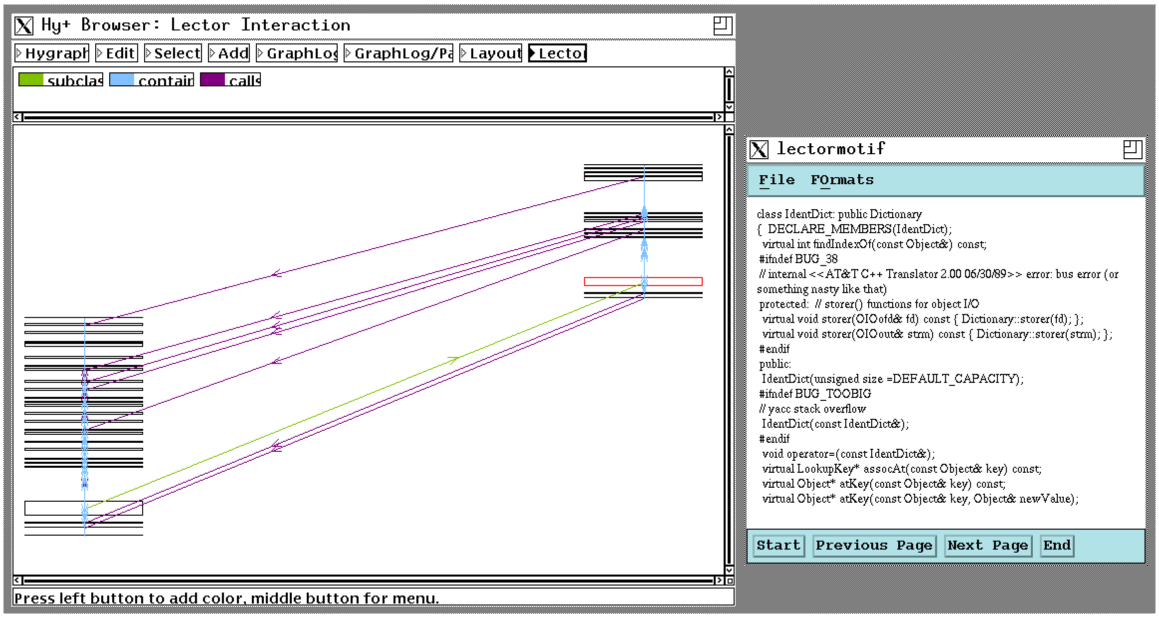

Figure 5: Synchronized Graphical and Textual Browsing of Source Code.

6 Tool Integration

The system has the ability to invoke external programs that, for instance, browse an object being represented by a node in one of the graphs displayed by the system. The Hy^+ visualizations can be used as overviews to locate information and then invoke third-party browsers to display the contents associated with the relevant objects. An obvious advantage of this approach over a purely navigational one, is the ability to use the convenience and expressive power of GraphLog patterns to retrieve the objects of interest, instead of attempting an often impractical brute force search. Of course, this is in addition of the use of Hy^+ to generate as many specifically tailored overviews as needed.

Figure 5 has an example of such an integration. To the right of a specialized hygraph browser there is a Lector[1] [Ray92] window that displays the source code associated with the object selected in the browser (the code for class('IdentDict')). The display synchronization works both ways: when the user changes the page of source code displayed by Lector the object selected in the Hy^+ browser adjusts correspondingly. Furthermore, the query evaluation component has been extended to handle a mixture of traditional and textual queries. The latter kind of queries are handled by the PAT Text Searching Engine [Gon87], which has been integrated as an additional query processor backend.

Acknowledgements

We would like to thank the other members of the Hy^+ development group for their contributions to the system: Frank Eigler, Yanni Jew, Masum Hasan, Emanuel Noik, Dimitra Vista, and Annie

[1] Lector and Pat are trademarks of Open Text Systems Inc.

Yeung. We are also grateful to Pat Finnigan, Shahram Javey, Willard Korfhage, Spiros Mancoridis, Darrell Raymond, Arthur Ryman, and Jacob Slonim.

The Hy$^+$ project is supported by the Natural Sciences and Engineering Research Council of Canada, the Information Technology Research Centre, the Institute for Robotics and Intelligent Systems, and the IBM Canada Centre for Advanced Studies. The first author was supported by an IBM Canada Research Fellowship and by PEDECIBA (Uruguay).

References

[CH93] Mariano Consens and Masum Hasan. Supporting network management through declaratively specified data visualizations. In *Proceedings of the Third IFIP/IEEE International Symposium on Integrated Network Management*, pages 725–738. IFIP Transactions C-12, Elsevier North-Holland, 1993.

[CHM93] Mariano Consens, Masum Hasan, and Alberto Mendelzon. Debugging distributed programs by visualizing and querying event traces. Abstract presented at the ACM/ONR Workshop on Parallel and Distributed Debugging, 1993.

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

[CM90a] Mariano Consens and Alberto Mendelzon. The G$^+$/GraphLog visual query system. In *Proceedings of the ACM-SIGMOD 1990 Annual Conference on Management of Data*, page 388, 1990. Video presentation summary.

[CM90b] Mariano Consens and Alberto 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 Consens and Alberto 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. A revised version has been accepted for publication in TCS.

[CMR92] Mariano Consens, Alberto Mendelzon, and Arthur Ryman. Visualizing and querying software structures. In *14th. Intl. Conference on Software Engineering*, pages 138–156, 1992.

[CMV93] Mariano Consens, Alberto Mendelzon, and Dimitra Vista. Deductive database support for data visualization. Technical Report CSRI-285, University of Toronto, 1993. (In Declarative Database Visualization: recent papers from the Hy+/GraphLog project, pages 51-66).

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

[Con92] Mariano P. Consens. Visual manipulations of database visualizations. PhD Thesis Proposal, 1992.

[Eig93] Frank Ch. Eigler. GXF: A Graph Exchange Format. Technical Report CSRI-285, University of Toronto, 1993. (In Declarative Database Visualization: recent papers from the Hy+/GraphLog project, pages 91-107).

[Fuk91] Milan Fukar. Translating GraphLog into Prolog. Technical report, Center for Advanced Studies IBM Canada Limited, October 1991.

[Gon87] Gaston Gonnet. PAT 3.1: An efficient text searching system. Technical report, UW Centre for the New OED, University of Waterloo, 1987.

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

[Noi93] E.G. Noik. Graphite: A suite of hygraph visualization utilities. Technical Report CSRI-285, University of Toronto, 1993. (In Declarative Database Visualization: recent papers from the Hy+/GraphLog project, pages 108-126).

[NT89] Shamim Naqvi and Shalom Tsur. *A logical language for data and knowledge bases*. Computer Science Press, New York, 1989.

[Ram92] R. Ramakrishnan, D. Srivastava and S. Sudarshan. Coral: Control, relations and logic. In *Proc. Intl. Conference on Very Large Data Bases*, 1992.

[Ray92] Darrell Raymond. Flexible text display with lector. *IEEE Computer*, 28(8):49–60, 1992.

[Tom89] Frank Tompa. A data model for flexible hypertext database systems. *ACM Transactions on Office Information Systems*, 7(1):85–100, 1989.