Hy+ User's Manual

May 3, 1993

Hy+ programs developed by:

Mariano P. Consens, Frank Ch. Eigler, Sergio R. Faria,
Masum Z. Hasan, Yanni K. Jew, Christine N. Knight,
Alberto O. Mendelzon, Carlos G. Mendioroz, Manny Noik,
Toomas Toomsalu, Dimitra Vista, Annie Yeung

Manual written by:

Frank Ch. Eigler
fche@db.toronto.edu

Contents

1 Introduction **1**

2 Hygraphs **1**

3 Starting and Stopping Hy+ **2**

4 The Browser **3**
4.1 Visualization . . . 4
4.2 Browser modes . . . 7
4.2.1 Hygraph . . . 7
4.2.2 Select . . . 9
4.2.3 Add . . . 12
4.2.4 GraphLog . . . 13
4.3 Label editing mode . . . 15

5 The Overview Browser **16**

6 Hy+ Tasks **18**
6.1 How to create a new hygraph . . . 18
6.2 How to load an existing hygraph from a disk file . . . 18
6.3 How to save a hygraph to a disk file . . . 18
6.4 How to close a hygraph browser . . . 18
6.5 How to layout a hygraph . . . 19
6.6 How to query a hygraph database . . . 19

A The Palette Editor **19**

B The Icon Map Editor **21**

List of Figures

1 Sample startup display of Hy+showing hygraph file browser popup menu ready to load a hygraph 3
2 Basic Hy+ hygraph browser, ready for interaction . . . 4
3 Hy+ display of a node, a blob, and several contained nodes. . . . 5
4 Hy+ display of several edges and nodes. . . . 6
5 More complicated Hy+ display, including several blobs in a node. . . . 6
6 Mapping from schematic hygraph elements to relational predicates . . . 13
7 Compound GraphLog query, in a browser. . . . 15
8 Basic Hy+ overview browser. . . . 17
9 Palette editor spawned as a separate window . . . 20
10 Color picker, with highlighted entries also used in the associated palette editor . . . 21
11 Functor icon map editor spawned as a separate window . . . 21

1 Introduction

Hy+ is a system for data visualizing and querying. It runs under the ParcPlace ObjectWorks Smalltalk (version 4.1) environment on several UNIX workstations running X-Windows.

This manual is intended to introduce new users of the Hy+ system to the program's capabilities and use. In order to limit the scope of this paper, we assume a certain familiarity with the plain Smalltalk environment, in particular these aspects:

- the concept of a Smalltalk virtual *image*,
- the process of executing and halting an image,
- the basic graphical user interface provided: multiple graphical windows, pop-up menus, text editors, the file/directory browser, interactive workspaces, mouse operations (clicking, dragging).

In order to become familiar with the above aspects, please see the ParcPlace user's manual for Smalltalk. Further, we use several concepts from database theory that are defined only in context of Hy+, for example *databases*, *queries*, *predicates*, *graphs*, *vertices*, *edges*.

The structure of this document is as follows. Section 2 describes the Hy+ data model, *hygraphs*, in an informal way. Section 3 gives instructions on how to start the Hy+ application and to create interactive visualization browsers. Section 4 describes in detail the functionality available within a browser. Such functionality includes editing, querying, and layout of hygraphs. Section 5 describes a non-interactive but quick hygraph visualization window called an *overview*. Section 6 gives an overview of the steps needed in order to accomplish various other Hy+ tasks, in terms of the actions defined in the previous sections.

2 Hygraphs

Hy+ deals with data in the form of a mathematical object called a *hygraph*. A hygraph is an extension of a *graph* because in addition to a set of vertices (or nodes) and directed edges (or arcs), a hygraph also includes a set of *blobs* (a type of nestable container). Informally speaking, in a hygraph, edges connect vertices as usual, but vertices are no longer necessarily atomic objects, as they may "contain" zero or more blobs, which in turn may "contain" zero or more nodes (and so on). This grouping structure allows recursive decomposition of a large graph into hierarchical components, much like Harel's formulation of *higraphs* in [Har88], but is more appropriate for visualization of predicate-based relational databases (see [Con92] for a formal definition).

A hygraph can be obtained from a relational database using conversion tools. A relational database written in the form of Prolog *facts* (see [You88]), for example, can be used directly by Hy+. Since Hy+ attaches a large amount of visualization-related information to hygraphs, we have developed several file storage methods to allow storage of such information apart from a particular Smalltalk image, e.g., GXF [Eig93], a format useful for external processing.

3 Starting and Stopping Hy+

To start using Hy+, a Smalltalk image that contains the Hy+ application must be invoked. First, start X-Windows as is usual on your workstation. Assuming you have installed the Hy+-related files into the directory `/home/hy+`, and the ObjectWorks Smalltalk 4.1 distribution for your computer into `/usr/smalltalk`, execute the following commands from a UNIX shell:

```
prompt % cd /home/hy+
prompt % ls *.im
        hplus.im
prompt % /usr/smalltalk/bin/st80 hplus.im
```

This sequence should start a Smalltalk virtual machine interpreter, and should result in the creation of a set of windows similar to that of figure 1. Some of these windows belong to plain Smalltalk, and some are managed by Hy+. The one of most interest is the window labeled Hy+ Hygraph Visualization System, which is a normal Smalltalk *workspace*. This window contains several Smalltalk expressions that may be executed (by highlighting the command of interest and selecting doit from the middle-button menu). Each of these expressions causes some Hy+ action to take place, some of which result in the creation of a new hygraph *browser*.

A browser is the primary user interface to a particular hygraph, and is the "place" where all manipulation and querying facilities of Hy+ are invoked from. There can exist several independent browsers on different hygraphs, and even multiple dependent browsers on a single "logical" hygraph (but more on this later). To create a new browser on a new empty hygraph (which the user can later build piecemeal), find the line in the Hy+ workspace that looks like

```
HygraphBrowser openOn: Hygraph new.
```

and execute it. On the other hand, to load a hygraph from a disk file and to open a browser on that hygraph, you may find it useful to use a file browser in order to locate the file. To open a file browser, find and execute the line in the Hy+ workspace that looks like

```
HygraphFileBrowser openOn: 'Data/*'.
```

Figure 1 shows an open Hy+ file browser. Exactly like the standard Smalltalk file browser (as available from the Launcher), it lets the user navigate and manipulate files and directories. When a file is selected in this file browser's middle panel, an extra browse menu option is available. If this command is invoked, Hy+ attempts to read a hygraph from the selected file and open a *browser* on that hygraph. An extra set database menu option is also available which loads a hygraph from the selected file (just as before), but rather than opening a browser, Hy+ sets that hygraph as a *database* for subsequent queries (see section 4.2.4). This is most useful for large hygraphs which would be too slow to visualize in their entirity.

Since Hy+ supports several file formats for the storage of hygraphs on disk, the program needs to identify the format for each particular file. It does this by identifying the file name's extension (".gxf", ".prolog", etc.), or if it is unable to match any known extensions, Hy+ opens a dialog from where the

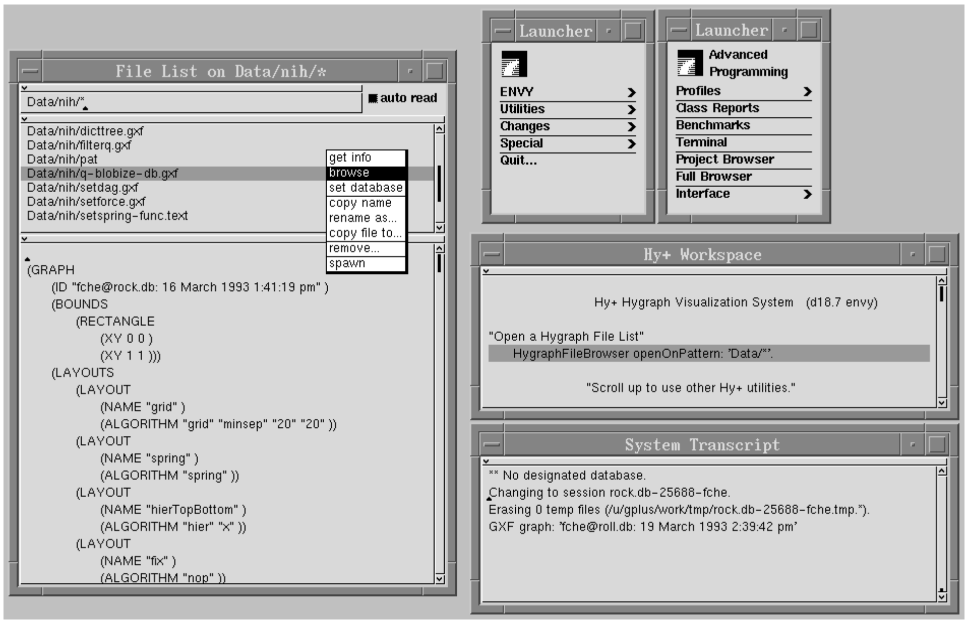


Figure 1: Sample startup display of Hy+showing hygraph file browser popup menu ready to load a hygraph

user must pick one of the known formats. If Hy+ cannot parse the file in the desired format, a blank browser opens (and an error message possibly appears). After a successful read operation, however, a browser opens on the desired hygraph, as in figure 2.[1] The following sections describe what facilities are available in the browser.

To stop Hy+, follow the usual procedure of selecting "Quit" from the Launcher, and if desired, saving the running Hy+ image.

4 The Browser

As shown in figure 2, a Hy+ browser consists of four main parts. At the top is a series of boolean buttons, one of which may be selected at a time and determines the interaction *mode* of the browser. Just below these buttons, an optional editor resides for the predicate color map used by this browser (see section 2). This editor is simple to use and displays instructions in the message line when the mouse pointer is within its bounds. In the middle of a browser is the hygraph *view*, which displays the interactive visualization of a hygraph. At the bottom is a *message line*, which contains various status

[1] Actually, since Hy+ is rather sluggish when dealing with large hygraphs, the file browser offers to open an *overview* rather than a browser on a large hygraph just read. Overviews are non-interactive but very quick representations of hygraphs, and are described further in section 5.

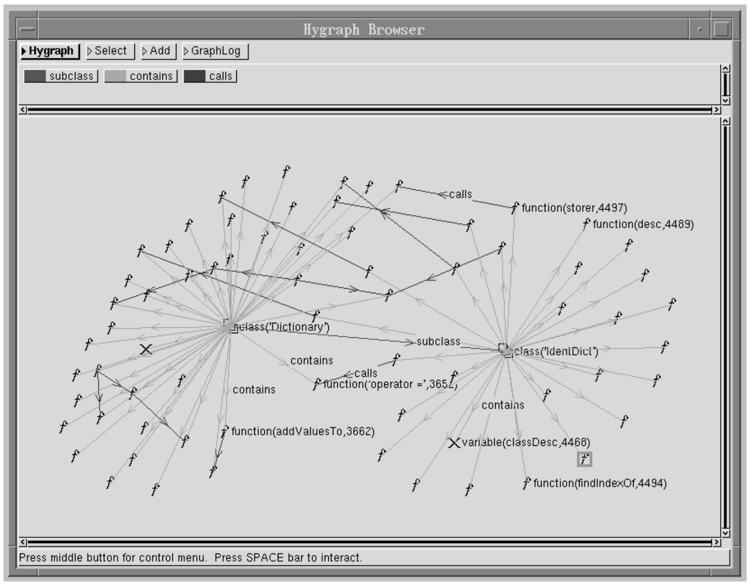


Figure 2: Basic Hy+ hygraph browser, ready for interaction

messages from Hy+ and supplements the usual Smalltalk cursor animation while lengthy operations are in progress. Error messages are also displayed in the message line.

4.1 Visualization

A Hy+ hygraph browser displays hygraphs with considerable flexibility. The basic display consists of a composite representation of each node, edge, and blob, as described below.

A node visualization consists of an optional textual "label" part and a small graphical node icon. The label may be displayed in one of several "styles", one of which removes the label entirely to reduce hygraph clutter. Hy+ chooses a node icon from about 40 available, considering the first (optional) functor component of the node's label. For example, a node labeled `city('Toronto')` will have an icon related to the `city` functor, if one is pre-drawn in Hy+. A node labeled without a functor (or with an unknown functor) is given an icon which is a small solid dot. This composite node visualization is drawn *contained* either at the "top level" (if the node is not contained in any blobs), or within the blob(s) it is contained in. If the node contains one or more blobs, an additional component appears, which is a larger rectangular region into which its contained blobs will be drawn. The size of this region is editable in the hygraph browser. If the node is contained by more than one blob, a "shadow" copy is added into the second and subsequent blobs, to keep the containment structure strictly hierarchical.

An edge visualization consists of a segmented polyline connecting its from-node to its to-node (with an arrow-head to indicate direction), and an optional textual label component. A circular edge (one with the same from- and to-nodes) is drawn as a circle instead of a polyline. Hy+ chooses a

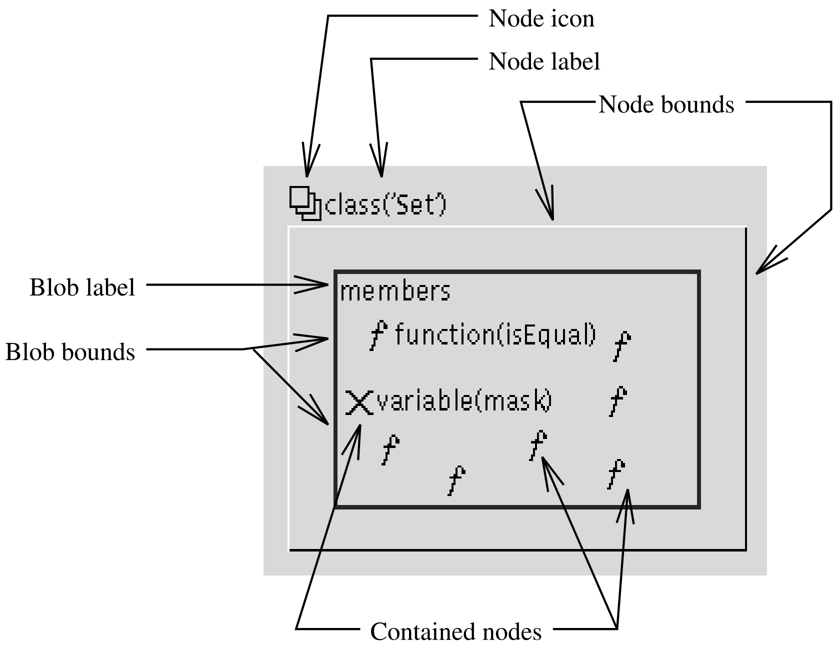


Figure 3: Hy+ display of a node, a blob, and several contained nodes.

color for the edge polyline or circle by finding a unique predicate name in the textual label (if any), and looking this name up in a *predicate color map* local to the hygraph browser. For example, an edge labeled `flight('Air Canada')` will pick the color related to the `flight` name (or a plain default gray if the predicate is a new one). A distinguished edge is shown thicker.

A blob visualization consists of an optional textual label component and a bounding rectangle which contains one or more node visualizations. Since a blob represents a relation much like an edge, its border is color-coded according to the label predicate. A distinguished blob is shown by a thicker border.

Moreover, the contents of nodes and blobs can be *hidden* selectively. Such hiding replaces the normally "transparent" region bounding contained elements by a colored "opaque" rectangle, which does not contain any detail. (The label of a blob or a node with hidden contents is still shown according to its "label style".) An issue arises about how to show edges that lead to nodes that are contained in some blob whose contents (thus the node) have been hidden. Hy+ can indicate the presence of such edges in several ways, including "not at all", by a few representative "dummy" edges that are drawn between the nearest visible (not hidden) ancestor nodes of the edges' endpoints. There is usually only one dummy edge between two nodes that represents *all* edges that have been hidden and belong between two hidden sub-nodes. Using Hy+ with most uninteresting parts hidden greatly increases its performance, so is recommended.

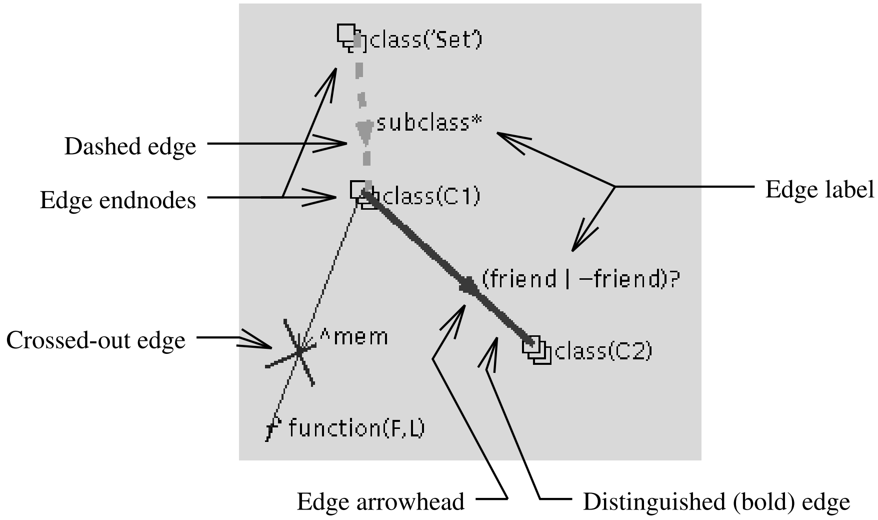


Figure 4: Hy+ display of several edges and nodes.

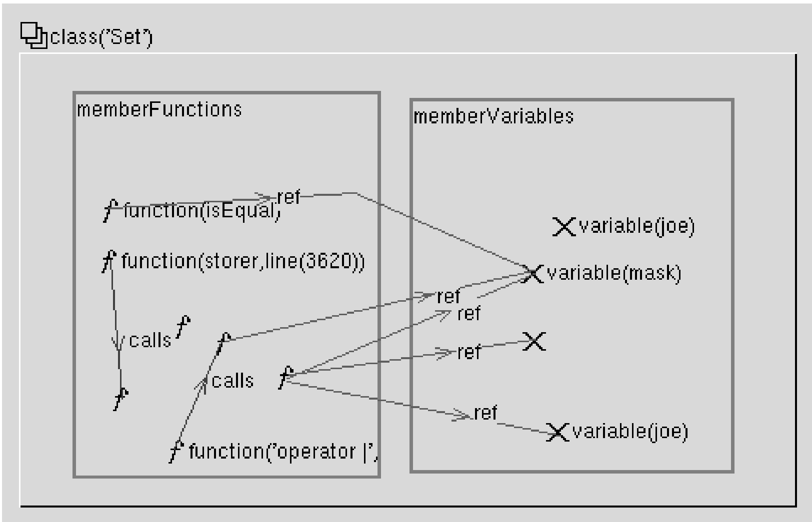


Figure 5: More complicated Hy+ display, including several blobs in a node.

4.2 Browser modes

By clicking on (highlighting) a browser *mode* button, the user selects one from several interactions available in the view part of the browser. These interactions differ in the ways that Hy+ responds to clicks of the left and middle mouse buttons. Usually the middle mouse button is used to activate a Smalltalk style pop-up menu where Hy+ commands are available. The following list outlines the scope of each mode selectable using the button list in the top pane of the browser. Further, there is a special sub-mode available in each main mode which allows the user to relabel objects in a hygraph. It is described in section 4.3.

Hygraph In this mode, the pop-up menu in the view invokes miscellaneous global functions, including hygraph importing/exporting, and view control.

Select In this mode, the user can interact with the hygraph in the view using a select-operate mechanism which includes cut and paste within or between hygraphs.

Add In this mode, the user can add new nodes, edges, and blobs to the hygraph being viewed.

GraphLog In this mode, the pop-up menu in the view controls the GraphLog query subsystem.

Layout In this mode, the pop-up menu in the view invokes the hygraph layout subsystem.

4.2.1 Hygraph

In Hygraph mode, the left mouse button is not used. The middle button invokes a pop-up menu with the following commands available:

file Opens a sub-menu related to file handling with the following entries:

- **new** Opens a new browser on a new empty hygraph.
- **open** Tells Hy+ to read a hygraph from a file. A dialog will ask for the file name. The file format will be guessed (if possible), as described in section 3.
- **merge** Copies the graph imported from an external file to a subrectangle in the browser. The user must select the file by entering its name in the usual file name dialog.
- **save** Writes the hygraph being browsed into a Unix file in one of several formats. A file name dialog appears. The user can select one of the “gxf”, “prolog”, “text” formats by adding it as an extension to the file name (e.g., `network.gxf`).

spawn Opens a sub-menu related to creating new browsers, which contains the following entries:

- **view** Opens a new browser on the *same* hygraph being browsed. That means that changes made in one browser will automatically be shown in the other browser as well.
- **copy** Opens a new browser on a new, independent *copy* of the hygraph being browsed.
- **overview** Opens a new overview browser on the *same* hygraph being browsed. See section 5 for its description.

layout Opens a sub-menu related to creating new layouts of hygraphs. These services are provided by an external program that performs layout on GXF-format hygraphs (see [Noi93] for some details). The sub-menu contains the following entries:

- **resize** Resizes the hygraph being browsed into a rubber-banded rectangle specified subsequently.
- **fit** Resizes and relocates the hygraph being browsed so that it more evenly covers the space available in the browser's view.
- **set top level** Selects the name of a layout method to assign to the top level of the hygraph being browsed.[2] A list of available layout names is shown to the user in a pop-up list from which she picks one. That list, at the time of this writing, includes the following: `circle, column, fix, grid, hierLeftRight, hierTopBottom, overlap, pack, random, row, spring, stack`.
- **execute layout** Executes the standard external layout program, passing it the previously selected layout names of the top-level and each hygraph element (see Select mode). After processing, laid-out hygraph is shown in a new hygraph browser.
- **generic layout** Executes an external layout program, just as above. However, the user must specify the location of the UNIX layout program in a dialog box.[3]

palette Opens a sub-menu that relates the browser's local predicate color map (or "palette") and the shared default palette that new browsers inherit. There is a simple editor for such palettes, described in appendix A.

- **set default** Copies the palette entries from the current browser's palette into the shared global default palette. The user will be asked to verify each entry that would overwrite and change one in the global palette.
- **get default** Copies the palette entries from the shared global default palette into the current browser's palette. The user will be asked to verify each entry that would overwrite and change one in the current palette.
- **remove unused** Shrinks the size of the local palette by removing every entry that refers to a predicate name that doesn't appear in the hygraph. (This operation is done automatically for large graphs at browser creation.)
- **show/hide** Toggles the appearance of the optional predicate color map editor above the hygraph *view* in the browser.
- **spawn** Creates a new color map editor on the *same* color map used by the current browser, but in an entirely separate window. A spawned palette editor is dependent on the original palette, just as several spawned hygraph browsers are dependent on the original hygraph.

[2] The top-level layout controls how nodes that are not contained by any blob are positioned relative to each other.

[3] The external program is expected to take a GXF format file on its standard input and return a GXF file on its standard output. See [Eig93].

icons Opens a sub-menu that relates the browser's local functor icon map and the shared default icon map that new browsers inherit. There is a simple editor for such palettes, described in appendix B.

- **set default** Copies the icon entries from the current browser's icon map into the shared global default icon map. The user will be asked to verify each entry that would overwrite and change one in the global icon map.
- **get default** Copies the icon entries from the shared global default icon map into the current browser's icon map. The user will be asked to verify each entry that would overwrite and change one in the current icon map.
- **remove unused** Shrinks the size of the local icon map by removing every entry that refers to a functor name that doesn't appear in the hygraph.
- **spawn** Creates a new icon map editor on the *same* icon map used by the current browser, but in an entirely separate window. A spawned icon map editor is dependent on the original icon map, just as several spawned hygraph browsers are dependent on the original hygraph.

hidden edges Opens a sub-menu related to the handling of hidden edges, that is, those edges whose endpoints are hidden in the hygraph view because an endpoint node is hidden (see section 2), but are nevertheless representable, as "dummy" edges between each hidden endpoint's nearest visible ancestor.

- **ignore** Makes hidden edges completely ignored – not displayed at all. This is the default setting for browsers.
- **empty** Represents hidden edges as single unlabelled dummy edges.
- **count** Represents hidden edges just as for empty, but also includes a count of the hidden edges represented by each dummy edge, for example as as `hidden(25)`.
- **detail** Represents hidden edges just as for empty, but also includes a detailed count of the represented hidden edges, itemized by predicate name. For example, if three edges labeled `ca`, `aa`, and `aa` are represented by a dummy edge, its label will be `hidden((ca(1),aa(2)))`.

4.2.2 Select

In select mode, the left mouse button is used to *select* or highlight hygraph elements one-by-one or as a group. The selection state of a hygraph node, edge, or blob, is shown by a yellow or black highlighting around its selection-click sensitive region, which is the node *icon*, the blob *label*, and the edge *arrow-head*. The set of all selected objects make up the *selection set*, which can then be used as the set of "operands" to commands available in the middle-button pop-up menu. Additionally, the left mouse button is used to drag (translate within their containers) selected objects.

In detail, the use of the left button in Select mode is as follows:

- Clicking on an unselected hygraph object or rubber-banding a rectangle *replaces* the selection set with the included objects.

- Holding down the SHIFT key and clicking on an unselected object or rubber-banding a rectangle *adds* the included objects to the selection set.
- Holding down the SHIFT key and clicking on a selected object toggles its selection state.
- Clicking on empty space (not on the selection region of hygraph object) clears the selection set.
- Dragging on a selected node or blob, enters *drag* mode, where all selected nodes and blobs are translated simultaneously (within their containers). Only the selected objects' outlines are shown during the dragging operation, for speed.

The pop-up menu in select mode has many entries, of which some entries may be disabled (shadowed or crossed-out) if not available in the context of the current selection set. For example, operations that are only defined when *single* objects are selected will be disabled when several objects are selected. The entries in the pop-up menu are:

select Opens a sub-menu which includes commands to add to the selection set an entire class of objects. This is much quicker than using many individual clicks. The entries in the sub-menu are:

everything Selects every node, edge, and blob.

nodes Selects every node.

blobs Selects every blob.

edges Selects every edge.

unselect Operates just as the select set of options, but subtracts from the selection set rather than adds.

copy Copies a subgraph induced by the selected objects to a "clipboard" for later pasting into some browser. The induced subgraph is the selection set, plus attached nodes for each selected edge, plus the containing node for each selected blob.

cut Copies a subgraph to a clipboard just as for copy, but then removes the selected objects from the hygraph.

paste Copies the subgraph contained in the clipboard (if any) to a subrectangle in the browser. The user must rubber-band a rectangle into which the subgraph is pasted.

remove Removes all selected objects, *permanently*, after the user verifies this intention in a dialog box.

nodes Opens a sub-menu containing commands that operate on those selected nodes that contain one or more blobs and thus have a blob bounding regions.

resize (This command is available only when a single node is selected.) It resizes the selected node by rubber-banding its blob-containing bounding rectangle.

- **shrink** Shrinks all selected nodes by some default factor.
- **expand** Expands all selected nodes by the same default factor, constrained to remain within their containing blobs, if any.

blobs Opens a sub-menu containing commands that operate on selected blobs.

- **resize** (This command is available only when a single blob is selected.) Resizes the single selected blob by rubber-banding its bounding rectangle within its containing node.
- **shrink** Shrinks all selected blobs by some default factor.
- **expand** Expands all selected blobs by the same default factor, within their containing nodes.
- **associate** Finds all selected blobs and nodes that have been dragged somehow on top of each other to *appear* contained in one another (by overlapping). It then adds the necessary containment relationships to the hygraph to really *make* these objects be contained in one another. This action never breaks containment hierarchy, so never causes node duplication.
- **dissociate** Removes any containment relationship between selected nodes and blobs while keeping the location of the affected objects unchanged.

contents Opens a sub-menu that operates on the contents (the set of contained objects) of selected nodes and blobs.

- **hide** Hides the contents of the selected nodes and blobs, and replaces the display of contained blobs and nodes by colored solid rectangles. Edges with end-points which have been hidden are optionally displayed according to the last setting of the hidden edges entry in the hygraph mode pop-up menu; see section 2.
- **hide recursive** Just as above, but tags as "hidden" all contents of selected objects, recursively.
- **show** Removes the "hidden" tags from the selected objects, thus displays their immediate contents (but the contents of the newly-shown objects may still be hidden).
- **show recursive** Removes the "hidden" tags from all contents of selected objects, recursively. This will force the display of all children of the selected objects (and all children's children, etc.).
- **fit** Attempts to rescale the contents of selected nodes and blobs so the contents make better use of the space available in their respective node- or blob-bounding regions.

edges Opens a pop-up menu that contains commands that operate on selected edges, which contains the following entries:

- **edit middlepoints** Allow the user to modify the middle-points of an edge arrow, similarly to the procedure of edge creation in add mode, described in section 4.2.3.
- **reconnect to** (This command is available only when a single node and some edges are selected.) Changes the to-node of every selected edge to the single selected node.
- **reconnect from** (This command is available only when a single node and some edges are selected.) Changes the from-node of every selected edge to the single selected node.

label style Opens a sub-menu which relates to the label style of selected objects, and contains the following entries:

- **none** Hides the label component of each selected object. The effects of blob and edge predicate color coding and node functor mapping remain unchanged.
- **simple** Sets the label component of each selected object to be a plain text line containing each's entire label.
- **editable** Sets the label component of each selected object to be a small text editor which can be used in the label-editing sub-mode (see section 4.3) to quickly relabel the selected objects.

distinguish Opens a sub-menu related to changing the "distinguished" status of selected objects, and contains the following entries:

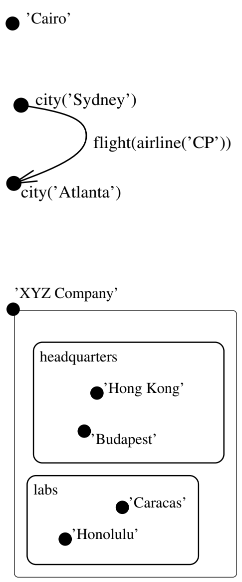
- **all** Distinguishes all selected objects.
- **none** Undistinguishes all selected objects.
- **toggle** Distinguishes all undistinguished selected objects, and vice versa.

set layout Assigns a layout method to all selected objects. A list of available method names is shown to the user in a pop-up list from which she picks one. When a layout is assigned to a node or a blob, the contained blobs and nodes of each will be accordingly laid out relative to each other when the layout is later executed using the Hygraph mode's layout menu.

4.2.3 Add

Add mode uses the left and middle mouse buttons to *add* new nodes, edges, and blobs to the hygraph being browsed. These new elements are initially labeled with the empty string, but that can be changed in the label editing sub-mode (see section 4.3). The behavior of Hy+ in Add mode is as follows:

- Clicking the left mouse button on empty space creates a node not contained in any blob.
- Clicking the left mouse button on space within the bounds of a blob creates a node that is contained in that blob.
- Clicking the left mouse button on a node icon starts the process for the creation of an edge between two nodes (or a circular edge around one node). This process consists of (optionally) clicking on empty space to add middle-points for the segments of the edge, then clicking on the icon of the "target" node (see instructions in the message line). The edge segments can be changed later, within the select mode in the edges menu using the edit middlepoints menu option.
- Clicking the middle mouse button on the icon of a node without contained blobs creates a new blob contained by that node, sized according to a rubber-banded rectangle. All nodes that lie within the selected rectangle that are siblings of the new parent node are made to be contained in the new blob.



An isolated node: → `vertex('Cairo').`

An edge: → `flight(city('Sidney'),city('Atlanta'),airline('CP')).`

Two blobs: →

```
headquarters('XYZ Company','Hong Kong').
headquarters('XYZ Company','Budapest').
labs('XYZ Conpany','Honolulu').
labs('XYZ Company','Caracas').
```

Figure 6: Mapping from schematic hygraph elements to relational predicates

- Dragging the middle mouse button within the bounds of a node that already has some blobs creates a new blob contained by that node, sized according to the rubber-banded rectangle. (There may exist several overlapping blobs contained by that node.)

4.2.4 GraphLog

GraphLog mode allows the user to pose queries on a hygraph database in the visual GraphLog query language (defined in [Con89]). Here, a hygraph database is treated like a collection of *predicates* on which deductive rules can be defined using the query language. A user of Hy+ can use several GraphLog subqueries to *define* new relations, to *show* a subset of (new and old) relations, then to *hide* a subset of the remaining relations. Hy+ computes the relations resulting from a chain of queries, then opens a browser on a new hygraph which corresponds visually to both the query result and the database, either laid out just as the database was, or by performing an explicit layout on the result. Alternatively, Hy+ can *select* the final matched subset of relations in the original database browser (if any). Figure 6 indicates how hygraph objects are mapped to relational predicates.

All parts of a query, i.e., the definition, show, hide, and layout sub-queries, are edited in a single normal hygraph that can be exported and imported, just as databases can. To separate the sub-queries, they are enclosed in specially named blobs (each in a dummy top-level node), as for example in figure 7. In particular, the various GraphLog subquery blob names are:

defineGraphLog This subquery consists of the definition of a single new derived relation. The new relation is represented by the single *distinguished* edge, node, or blob in the subquery. The syntax and semantics of this subquery are a superset of those defined in [Con89]. Among other subqueries, definitions are always evaluated first. Separate defineGraphLog sub-queries can refer to the new predicates defined by other define sub-queries (as long as the definitions are not circular, and thus can be evaluated in some order).

showGraphLog This subquery consists of the selection of a subset from the predicates in the query database plus any newly derived relations from defineGraphLog subqueries. The predicates are selected by pattern matching according to the same GraphLog rules used in the defineGraphLog subqueries, except, any subset of the objects here can be *distinguished*. Those objects that form part of a GraphLog-style pattern match, and correspond to distinguished objects in the query, are *selected* to be included in the result. Separate showGraphLog queries result in the *union* of their individual results. Since a composite query's *show* subqueries are evaluated (in some arbitrary order) *after* all *define* queries, thus can refer to these newly defined predicates as well as those in the original database.

hideGraphLog This subquery consists of the subtraction of predicates from the cumulative results of defineGraphLog and showGraphLog queries. The semantics of this subquery are exactly those of showGraphLog, except matched distinguished predicates are subtracted (if present) from the cumulative results. *Hide* queries are evaluated (in some arbitrary order) *after* all *define* queries, and can refer to these newly defined predicates as well as those in the database.

layoutGraphLog(NAME) The presence of such a subquery indicates that a layout should be performed on the final query result before it is displayed. This subquery consists of a matching query just like showGraphLog except that the edges, blobs, and nodes corresponding to the matched distinguished predicates are assigned the layout name `NAME` for the final layout. (The word `NAME` should be replaced with a known layout method name: one of those available in the Select mode's set layout list.) If the subquery contains no nodes and edges (i.e., consists of only the layoutGraphLog blob and its container), then the hygraph's top-level is assigned the method `NAME`. *Layout* queries are evaluated (in some arbitrary order) *after* all *define*, *show*, and *hide* queries, and can refer to newly defined predicates as well as those in the database.

The query process is started from the middle-button menu entries in GraphLog mode, which has the following entries:

set database Sets the hygraph being browsed to be the database on which subsequent queries can be posed. Such a hygraph must represent a proper database consisting of *ground terms*, that is, edges, nodes, and blobs must be labeled, using only constants (i.e., no variables). If the hygraph is not acceptable, an error results and a diagnostic is shown in the message line of the browser.

execute query Evaluates the compound query in the hygraph being browsed, on the database previously set (in some other browser). As a special case, Hy+ will infer that the entire query hygraph represents a *show* query if there is no showGraphLog query blob present. If all subqueries complete successfully, a new browser will appear containing the result hygraph (optionally laid out if a layoutGraphLog subquery is present).

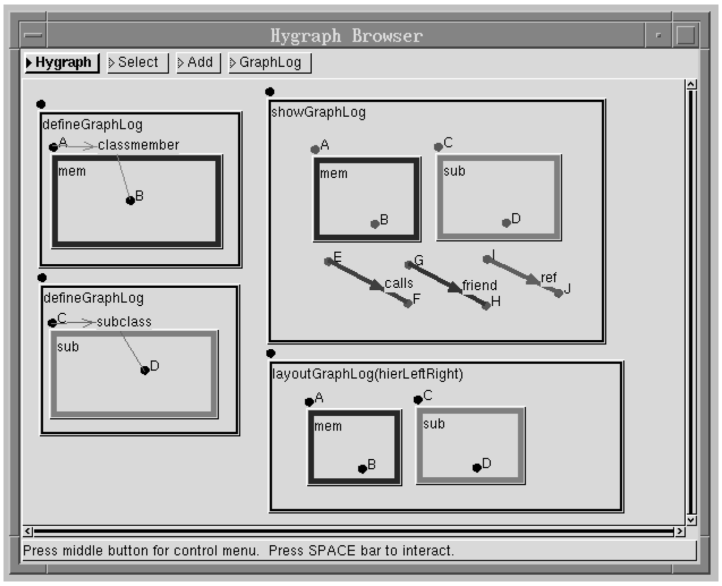


Figure 7: Compound GraphLog query, in a browser.

execute & select Evaluates the compound query in the hygraph being browsed, as before. However, those matched relations remaining after the *define*, *show* and *hide* subqueries, are *selected* in the the browser(s) containing the query's database. Any layoutGraphLog subqueries are ignored, and no layout is performed.

There is an additional consideration when new predicates are defined and shown in the result, which is that in order for these new predicates to be displayed in a hygraph, they must be given a shape. New predicates displayed as edges pose no problem and will be displayed as a straight edge between its endpoints. New predicates displayed as nodes will be given a random location. New predicates displayed as blobs will be given small (or zero) extents attached to their container nodes. In any case, it is likely that the result may contain initially invisible information. Therefore, for queries which define new blobs or nodes, the user may prefer to perform a subsequent layout on the result to give these new objects a better location. The easiest way is to include an empty layoutGraphLog subquery that gives the top-level of the result a reasonable layout, say "spring".

4.3 Label editing mode

Label editing mode is a special browser sub-mode, available from all modes discussed above. Its purpose is to allow the user to relabel edges, nodes, and blobs in a hygraph using familiar Smalltalk text editor windows, and to provide some integrated syntax checking. Syntax checking is useful because GraphLog queries and databases can consist of fairly intricately labeled hygraph elements, and without some automatic checking it would be hard to identify some "misspelled" elements that may have caused a query or a database to be rejected.[4]

Entering and exiting the label editing sub-mode is accomplished by pressing the SPACE BAR on the keyboard, while the mouse pointer is located in the view area of the browser. The message line will reflect this state change. At the time the browser enters this sub-mode, the labels of all *selected* objects in the hygraph are temporarily made *editable* (in addition to those whose labels are already editable due to previous **label style** menu commands in the **select** mode). The message line also displays an indication of the label editing mode.

While in the interact mode, the user may interact with any of the visible text editors by moving the mouse pointer within their bounds and using the usual text editor mouse/keyboard controls, e.g., cut/paste, selection, etc. To commit to a changed label for a hygraph element, the user presses the ENTER key. Hy+ checks the syntax of the label to ensure that it is either a valid constant term (for databases) or a valid expression (for queries). If the label is acceptable, the text editor reformats the label by removing unnecessary quoting and whitespace. Otherwise, the text editor just flashes.

To exit interact mode and return to the previous main mode, move the mouse pointer outside all text editor subwindows (but still within the browser view area), and press the SPACE BAR again. If any hygraph element labels were edited but not committed (by pressing the ENTER key), a confirmation dialog will appear for every such instance, to prevent accidental loss of data. When interact mode is exited, the labels of the selected hygraph elements will be returned to their states before interact mode was entered (that is, their label styles will be reset to "none", "simple", or remain "editable"). Another way to exit the interact sub-mode is to switch to another main mode by clicking on a new browser mode select button.

5 The Overview Browser

The overview browser is a special non-interactive view of a hygraph. It displays a quick but undetailed representation of a hygraph: each node is represented by a small rectangle; each edge by a small partial line segment between its endpoint nodes, or a solid straight line, or not at all; each blob is drawn as a thin colored rectangle are not at all. Such a simplified display is quite adequate to get an overall view of the entire hygraph, but without the overhead of opening a full browser. Figure 8 contains an overview browser of the same hygraph displayed in figure 2. Just like a full browser, an overview also has a message line at the bottom.

The middle mouse button invokes a pop-up menu in the overview, containing the following entries:

spawn browser Opens a new browser on the *same* hygraph being overviewed. That means that changes made in the browser will automatically be shown in the overview (and possibly other browsers and overviews).

[4] Labels in databases have more restrictive syntax requirements in that only constant terms may appear there. Unfortunately the syntax checking can not tell potential databases apart from queries, so Hy+ checks a label for obeying query syntax only (which includes database syntax). Therefore, a hygraph that has labels of "accepted" syntax may still be rejected when the user attempts to set it as a database.

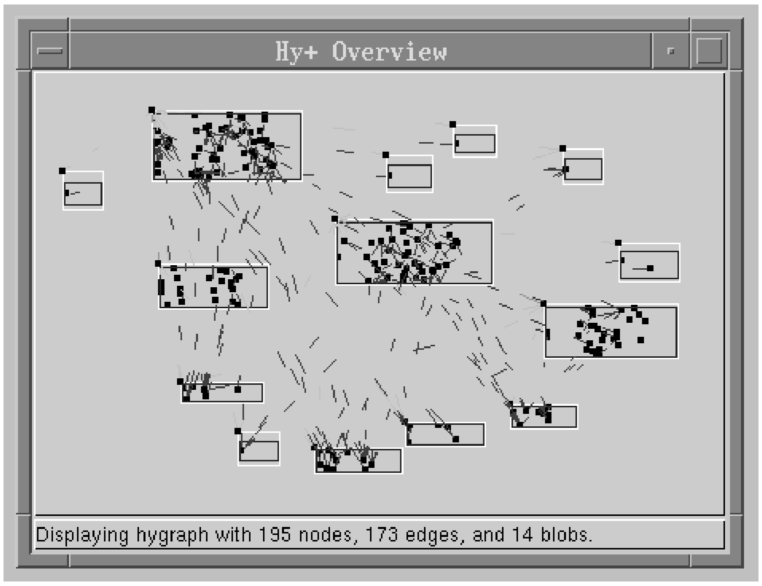


Figure 8: Basic Hy+ overview browser.

set database Sets the hygraph being overviewed to be the database on which subsequent queries can be posed. This operation is just like the set database option in the GraphLog mode of a browser.

file Opens a sub-menu related to file handling. This menu behaves just as the corresponding menu in Hygraph mode of a browser. The sub-menu contains the following entries:

- **import** Tells Hy+ to read a hygraph from a file.
- **export** Writes the hygraph being overviewed into a UNIX file.

layout Opens a sub-menu related to hygraph layout. This menu behaves just as the corresponding menu in Hygraph mode of a browser. The sub-menu contains the following entries:

- **set top level** Selects the name of a layout method to assign to the top level of the hygraph being browsed.
- **execute layout** Executes the standard external layout program.
- **generic layout** Executes a user-specified external layout program.

show edges Opens a sub-menu containing options that control how edges in the overviewed hygraph are to be drawn:

- **not at all** Hides edges.
- **fuzzy** Draws edges as partial line segments.

normal Draws edges as solid lines.

show blobs Opens a sub-menu containing options that control how blobs in the overviewed hygraph are to be drawn:

not at all Hides blobs.

normal Draws blobs as thin rectangles.

6 Hy+ Tasks

This section gives a point form outline to describe the most interesting user actions of Hy+, including the processes of layout of a hygraph, and querying a hygraph database. We refer to individual menu items and actions from related parts of section 4 and section 5, where the reader should look for descriptions of the individual steps.

6.1 How to create a new hygraph

1. **Create an empty hygraph browser.** Use the Hygraph mode's view - new option in some existing browser. Alternately, select and execute the appropriate expression from the Hy+ workspace as described in section 3.

2. **Add desired hygraph elements.** Use the Add mode in the new browser to create new nodes, edges, and blobs, and use the Select mode's cut/copy/paste operations to speed up building your hygraph.

6.2 How to load an existing hygraph from a disk file

1. If you know the file's name, and already have some Hy+ browser, it is quickest to use the Hygraph mode's file - import function. If not, then locate your desired file in a hygraph file browser, as described in section 3. In both cases, the file format should be guessed from the file name extension (if any), or should be picked from a dialog box (if file name has no extension).

6.3 How to save a hygraph to a disk file

1. Use the Hygraph mode's file - export option in the browser which contains the hygraph. Select a file-name as appropriate, and add the extension ".gxf" to specify use of the "GXF" file format (or "prolog" for the Prolog format, etc.). The GXF format stores enough information in the file to be able to recreate exactly the state of the hygraph when it was saved, including its layout, thus is recommended.

6.4 How to close a hygraph browser

1. Use the Smalltalk-standard right mouse button menu's close option to close the browser you do not want to keep. Hy+ deallocates unused resources (including hygraphs) automatically after

no further references exist, that is, after all browsers on a hygraph are closed, and it is not (or no longer) set as the database for the query evaluation subsystem.

6.5 How to layout a hygraph

1. Open a browser or an overview on a hygraph. Create a new hygraph, or load a hygraph from disk as described above.

2. Select desired layouts. In the Hygraph mode's layout menu, select a top-level layout using the set top level option. Select additional layouts for hygraph elements in the Select mode using the set layout option, if desired.

3. Execute layout. In the Hygraph mode's layout menu, select execute layout. An external program is called to perform the layout, then a new hygraph browser or an overview opens on the result.

6.6 How to query a hygraph database

1. Load and set database hygraph. Open a hygraph browser on your desired database, then select set database in the GraphLog mode. Alternately, use the set database option in a hygraph file browser (right under the browse option described in section 3) if you prefer not to open a browser on your database.

2. Pick query. Open a hygraph browser on your desired query (after loading it or creating it from scratch), including the sub-query grouping blobs described in section 4.2.4, if necessary. (Hy+ implies a single *show* query if there is no explicit showGraphLog sub-query present.)

3. Execute query. Use the GraphLog mode's execute query menu option in the browser containing the *query*. If the query is correct, after processing a new browser appears containing the query result. If the query contains an error, such as a reference to an unknown predicate, the message line will contain some indication of this.

A The Palette Editor

The predicate color map editor (or palette editor for short) allows the user to change the predicate name to color mapping used by hygraph browsers and overviews to draw edges and blobs. Instances of the editor can be embedded in hygraph browsers, or spawned into separate windows. Figure 9 shows a palette editor, browsing in this case the Hy+ default palette. (Opening a browser on this palette is an operation available in the Hy+ workspace – see section 3.) The message line at the bottom contains instructions about the effects of the mouse buttons.

Each predicate name is drawn with a rectangular sample of the color it is associated with. The editor supports adding and removing predicates from the palette, changing the display sort orders, and saving and loading the entire palette. The operations available depend on whether the mouse pointer is over a predicate name/color box or just over the blank background of the palette editor. Each

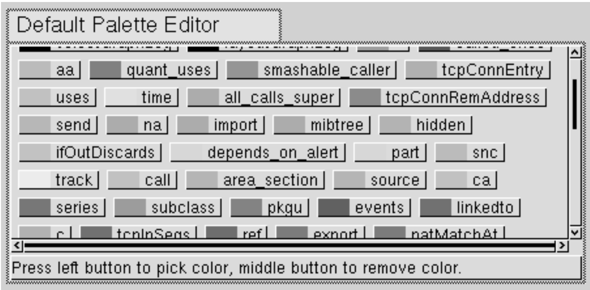


Figure 9: Palette editor spawned as a separate window

change propagates the new palette information to any dependent hygraph browsers immediately. The behavior of a Hy+ palette editor is specifically as follows:

- Clicking the left mouse button on empty space creates a new predicate to color mapping. The user enters the predicate name, then selects one of the available colors from a color picker (as in figure 10, see below).
- Clicking the left button on a predicate name/color box updates the color associated with that predicate name. The user picks a new color from the color picker.
- Clicking the middle button on empty space pops up a menu with the following options:

  load Loads and merges a previously saved palette from a UNIX file into the one being edited. The user specifies a file name in a dialog box.

  save Saves the palette being edited to an external UNIX file, whose name is also specified in a dialog box.

  sort Opens a sub-menu which controls in what order the predicate name/color boxes are ordered in the editor. Sorting is available by predicate name, and by color (“hue”, “luminance”, and “saturation”).

  direction Opens a sub-menu which controls which direction the sorted predicate name/color boxes are drawn to fill the editor window: top-to-bottom (“column”) or left-to-right (“row”).

  spawn Spawns a new palette editor on the same palette being edited. The new editor may be sorted and displayed differently, but is

- Clicking the middle button on a predicate name/color box removes that predicate (after a confirmation) from the palette.

The color picker (shown in figure 10) is a simple dialog from which the user can pick a color for a predicate in a palette editor. Each color slot in the rectangular grid is individually editable using a small interactive color cell editor when the middle mouse button is clicked over a rectangle. We leave it up to the user to experiment with the color cell editor!

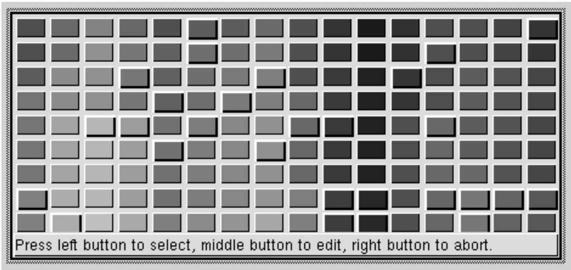


Figure 10: Color picker, with highlighted entries also used in the associated palette editor

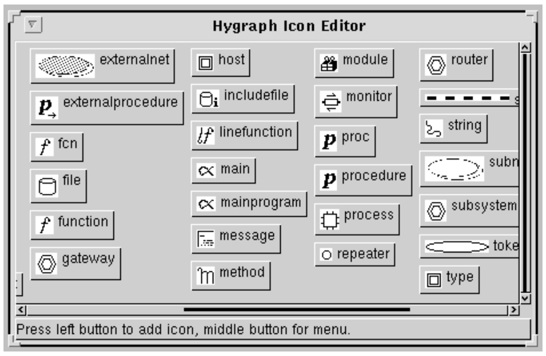


Figure 11: Functor icon map editor spawned as a separate window

B The Icon Map Editor

The functor icon map editor allows the user to change the functor name to icon mapping used by hygraph browsers to draw a distinctive icon for some nodes. Instances of this editor (as opposed to the palette editor) can only be spawned into separate windows. Figure 11 shows a icon map editor, browsing in this case the Hy+ default icon map. (Opening a browser on this icon map is an operation available in the Hy+ workspace – see section 3.) The message line at the bottom contains instructions about the effects of the mouse buttons.

Each functor name is drawn beside the picture of the icon it is associated with; entries in the icon map are always sorted alphabetically. The editor supports adding and removing functors from the icon map, and saving and loading of the entire icon map. The operations available depend on whether the mouse pointer is over a functor name/icon box or just over the blank background of the icon map editor. Unlike predicate color maps, changes in an icon map are not propagated immediately to dependent hygraph browsers, but rather, a new browser must be opened to see the node icons displayed differently. The behavior of a Hy+ icon map editor is specifically as follows:

- Clicking the left mouse button on empty space creates a new functor name to icon mapping. The user enters the functor name, then selects any region from the host X-Windows screen, which will then be imported into Hy+.
- Clicking the middle button on empty space pops up a menu with the following options:

  load Loads and merges a previously saved icon map from a UNIX file into the one being edited. The user specifies a file name in a dialog box.

  save Saves the icon map being edited to an external UNIX file, whose name is also specified in a dialog box.

  direction Opens a sub-menu which controls which direction the sorted functor name/icon boxes are drawn to fill the editor window: top-to-bottom ("column") or left-to-right ("row").

  spawn Spawns a new icon map editor on the same icon map being edited. The new editor may be sorted and displayed differently, but is

- Clicking the middle button on a functor name/icon box removes that functor (after a confirmation) from the icon map.

References

[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 manipulation of database visualizations. PhD Thesis Proposal, in preparation, 1992.

[Eig93] Frank Ch. Eigler. GXF: A Graph Exchange Format. CORDS White Paper, available for internet FTP at db.toronto.edu, January 1993.

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

[Noi93] E.G. Noik. Unix filters to draw and manipulate gxf graphs. University of Toronto Internal Document, February 1993.

[You88] Richard J. Young. *Practical Prolog*. Van Nostrand Reinhold, 1988.