MALUAR - A COMPUTATIONAL HYPERTEXT ENVIRONMENT

by

Tomás Isakowitz
Information Systems Department
Leonard N. Stern School of Business
New York University
New York, New York 10012

July 1992

Center for Research on Information Systems
Information Systems Department
Leonard N. Stern School of Business
New York University

Working Paper Series

STERN IS-92-21

Contents

1 Overview of the system

Maluar is a Hypertext system designed to handle documents and links between them. It provides a basic set of tools that can be used to create more elaborate Hypertext Environments.

The system is implemented in Allegro Common Lisp. It uses the Emacs-like FRED editor which is part of the Allegro Common Lisp programming environmnent.

The system contains two first class objects: Nodes and Links. The nodes are used to contain information and the links to establish relationships between nodes. A link is a reference from a region of one node to a region of another node. The system provides interactive functions to support link and node maintenance: creation, deletion and update.Browsing of Hyper-Documents is supported by link traversal. The system supports keyword handling. Whenever a keyword is found, a link is created from the occurrence of the keyword to its definition.

The system is Object-Oriented and has an open architecture. The Lisp funtions it provides cam be called by other Lisp programs. As an example of this, I have included with the system a program that identifies all Lisp functions occurring in a node and defines them as keywords. This way any other program that uses these Lisp functions will automatically have links to the functions definitions.

2 How to use Maluar

To start the system double click on the program "start.lisp" which will launch Allegro Common Lisp and start the system. Currently, the system is to be run in interpreter mode. Messages will appear on the screen as the different components of the system are loaded. The menu-bar will be updated with new options and the Messages window will appear with the following contents:

```
   ***************************************************************************

Welcome to Maluar.

   You can start working by selecting the appropiate menues on the
menu bar.

   For a demo, open the "memo-jan-28" node in the DEMO folder.
***************************************************************************
```

The last four menues in the menu-bar pertain excuIsively to the Maluar system, the first five are part of the Allegro Common Lisp environment. Here is a list of the actions that are supported by the system:

2.1 NODES

All node operations are accessible from the "nodes" menu.
Menu Items:

- **New Node**: To create a new node. A dialog box will prompt you for the name of the new node.
- **Open Node**: To access an already existing node.
- **Save Node**: To save the contents fo a node (it doen't close the node).

- **Close Node**: Closes the window of where the node appears. If the node has not been saved, you are given the opportunity to do so.

2.2 WEBS

A collection of links is called a WEB. The system allows you to use different webs on the same set of nodes. This supports node content sharing among different users while keeping the links that each user defines separate. When the system starts, the links in the "node:WEB" file are loaded.
Menu Items:

- **Reset Web**: clears from memory all links currently in use (Caution: the links are not saved. You should also "un-highlight" the links before doing this.)
- **Merge a Web**: add all links in an existing Web to the current set of links.
- **Load a New Web**: clear the current setb of alinks and load all links in an existing Web.
(Caution: the links are not saved. You should also "un-highlight" the links before doing this.)
- **Save Web**: save the current set of links under the name last given to the Web.
- **Save Web as**: give a name ans save the links under that name.

2.3 LINKS

The system supports inteactive link maintenance and manipulation.
To create a link the following procedure should be followed:

1. select the source of the link,
2. choose "Set link source" from the LINKS menu,

3. move the mouse to the node where the destination of the link is to be placed,
4. either position the mouse or select a region to be the specific destination position of the link,
5. choose "Set link destination" from the LINKS menu, and
6. choose "Create link" from the LINKS menu.

NOTE: One can also perform the previous proceduer in reverse order, selecting first the end-point and then the source-point of a link.

To traverse a link: position the mouse within the link source (a button), and choose "Traverse Link" from the LINKS menu.
Menu Items:

- **Set link source**: define the origin of a link.
- **Set link destination**: define the destination of a link.
- **Delete links**: dekete all links in the highlighted region.
- **Highlight Links**: show all links in the current document by changing font.
- **Un-highlight links**: hide all links by changing the font back to plain text.
- **Traverse Link**: traverse the link (see above).

2.4 KEYWORDS

Certain words can be defined as keywords. This entails selecting a word and specifying its definition. The system will then recognize occurrences of these keywords and create links to teir definitions.

To create a keyword and its definition do as follows:

1. select the word to be defined as a keyword,

2. choose "Current Keyword to be defined" from the menu,
3. Move to the node where the definition for this keyword is to be placed
4. select the text that defines the keyword
5. choose "Set Current Keyword's Definition" from the menu.

Menu Items:

- **Current Keyword to be defined**: to define a keyword (see above).
- **Set Current Keyword's Definition**: to define a keyword (see above).
- **Recognize Keywords in this node**: will recognize and create appropriate links for each keyword occurrence in this node.
- **Make Lisp functions into Keywords**: this only runs on nodes which contain Lisp programs. It will recognize function definitions and make each such function into a keyword.

3 Architecture of the system

We use an Object-Oriented approach. There are three principal types of objects: Nodes, Links and Editor windows. The system provides maintenance for these objects. There are functions to create, delete and update each of these entities as well as some housekeeping. The system is loaded by running the program "start.lisp" which defines the directory from which the called was launched as the home directory for all file operations. This involves accessing programs as well as documents. This option supports portability. Copying the complete folder to another location will not affect it. The program "start.lisp" calls "hypersystem.lisp" to define global variables and load the rest of the code.

We proceed to describe each type of object.

3.1 Nodes

The code pertaining to nodes is stored in the file: HYPERMACS.LISP. The following hierarchy is for nodes.

```
         node
           |
           |
         file_node
           |
           |
fred_window_file_node
```

"file_node" nodes have files associated with them. The node contents are associated in these files.
"fred_window_file_node" nodes are "file_nodes" which are setup so that the text in the files will be accessible via FRED, the emacs-like editor. Whenever such a node is accessed, the editor is invoked to allow the user to inreact with the file contents. This is the most common node in this system.

Each node is named at creation time by passing a ":name" parameter. The system keeps a table of all active nodes during a session. This table is used to determine how the node is to be accessed. If the node is active, there has to be an active window associated with it, node acyivation consist of bringing this window to the front. If the window is not active, it is activated (in the case of file_nodes by reading the file contents into a new window).

3.1.1 file_Nodes

These are a subtype of nodes. There is a separation between nodes and their contents. Each node that has been created is entered into a table (in the form of a list called "existing_nodes_list."). This table associates with each node name the complete filename of the file where its contents are stored. The table is automatically updated whenever a node is created. Deletion when destroying a node is not yet implemented.

3.1.2 fred_window_file_node

These are the most commonly used nodes within the system. Their contents are to be manipulated via FRED, the emacs like editor upon which the user-interface of the system is based. Each "fred_window_file_node" node has a FRED window associated with it. Whenever such a node is opened, the window is displayed and the name of the node i entered into the "nodes" menu in the menu-bar to make all nodes accessible via the "nodes" menu. When closing such a node the node's name is removed from that menu.

3.2 Window objects

The "hyper_fred_window" object is defined as a subtype of the built-in *fred-window* object type. It inherits all the Emacs-like editing features from that objects. I have added onto it a set of hypertext features that support link manipulation. This is achieved by defining object specific functions such as character insertion, deletion, cutting and pasting which will take care of manipulating the links properly, and by also changing the effects of keyboard activity via the comtab concept present in FRED. I define a special comtab named: *hyper-comtab*. The objects of type hyper_fred_window have this special comtab associated with them. There are a number of link m anipulation activities defined as object functions for "hyper_fred_window"s, these are explained further on in the links section.

3.3 Node Addresses

This type of object tantamounts to "buttons" in other systems. It contains the infomation neccessary to anchor the links. A node_address consist of:

1. node_id which is the name of a node
2. start_pos: which is the starting position within that node and
3. end_pos: which is the ending position within that node.

3.4 Keywords

The system supports the concept of a "keyword". It associates with each keyword a definition. Keyword recognition routines will identify keyword occurrences in a given node and create a link connecting each keyword occurrence with its definition. Keywords are thus supported by a special type of link: keyword_link.

3.5 Links

The folowing is the hierarchy for links:

```
        nil
         |
         |
        link
         |
         |
    keyword_link
```

Links have a number, a source address, a destination address and a type. Keyword links in addition have information abaout the actual keyword they represent. The type of a link is to be used to identify different kinds of links. Currently, it is only used when determining the font with which to highlight a link in a document. This is done via a font_table defined by the function (font_spec link) as follows:

- Links created interactively are of type "menue" and are displayed as underlined.
- Links generatde by the system are of type "system" and are displayed shadow.
- Keyword links appear in boldface.
- Other links will appear in italics.

The system keeps track of all links currently in use in a global variable called "link_list". Whenever a link is created or deleted this variable is modified accodingly.

Whenever a WEB is saved, this list of links is written onto file. Whenever a WEB is loaded its contents replace those of "link_list", and whenever a WEB is merged, its contents are added to "link_list".

4 The system as a toolbox: functions

4.1 Functions on nodes

The system keeps track of the nodes which are active in the current session. When accessing a node, the system first checks wether that node is active. If it is, its window is brought to the front. If it is not active, a new window is opened and the node contents read into it. This is achieved via the following functions:

- add_to_active_node_base(node)
- remove_from_active_node_base(node)
- active_node (name): if there is a node named "name" which is active, it is rteturned.

Object Functions for opening and closing nodes are provided:

- nopen: adds the node to the list of active nodes.
- nclose: removes teh nodes from the list of active nodes.

4.1.1 File Nodes

When creating a new node, a filename is specified and the pair is added to table of existing nodes. This allows links to refer to a node name instead of a file name, thereby supporting portability (copying a file from one location to another one involves chaging the entry in the "existing nodes" table. One can also remove such a pair from the table. This is achieved with the following functions:

- add_to_existing_nodes_list(name fname), and
- remove_from_existing_nodes_list(name).
- existing_node (name): if "name" is in the list of existing nodes, the pair (name fname) is returned.

The table of existing nodes is kept on disk. The following functions are used to achieve that:

- read_existing_nodes_list (): loads the table from disk.
- write_existing_nodes_list (): writes the table onto disk.

4.1.2 fred_window_file_node

Recall that objects of type "fred_window_file_node" are file_nodes that have an EMACS-like window associated with them. The following object functions are provided for "fred_window_file_node".

- nopen: a window is opened (type hyper_fred_window) and is associated with this node, the name is added to the "nodes" menu.
- nclose: the window is closed, the node's name is removed from the "nodes" menu.
- fwsave: the contents of the node are saved onto the associated file.
- activate_yourself_at_position(position): makes the current node active placing the cursor at the given position.
- activate_yourself_at_current_position: makes the current node active leaving the cursor where it was before.

4.2 Window objects

The nodes of type fred_window_file_node have windows associated with them which are of type hyper_fred_window. The following functions are provided for this object type:

- get_selection_range: returns a pair consisting of the start and end of the current selection within the window.
- selection_length: returns the length of the current selection within the window.

- hyper-buffer-insert: inserts a stringin the buffer.
- window-close: closes the window. It also tells the owning node to close itself.

All text editing is performed in this type of window, thisa is achieved by using the built-in FRED functions. I augmented these functions to support link maintenance.

- ED-INSERT-CHAR (character): a "hyper_fred_window" object function that inserts a character. It calls "delete_all_links_with_source_affected_by_selection" in case that the user is typing over a selection (thereby erasing it) and then "move_links_over position 1" to correct the positioning of links.
- ED-DELETE-WITH-UNDO (start end &optional (save T)): a "hyper_fred_window" object function that deletes the text in the affected region and updates the position of all links which occur beyond the end of the selection.
- cut(): the "hyper_fred_window" object function version of normal cut, it deals with the links attached to the text being cut.
- copy(): the "hyper_fred_window" object function version of normal cut, it deals with the links attached to the text being copied. It calls "copy_links" to do this.
- paste(): the "hyper_fred_window" object function version of normal cut, it deals with the links attached to the text being pasted. It calls "move_links_over" to do this.
- copy_links (): the "hyper_fred_window" object function that copies all links within the current selection into a temporary list so that these can later be pasted onto another location.
- paste_links (): the "hyper_fred_window" object function that pastes the links attached to the text in the clipboard.
- hyper_buffer-delete (start end): a "hyper_fred_window" object function that deletes the text and associated links within the boundaries specified by start and end.

- hyper_buffer-insert (str pos): a "hyper_fred_window" object function that inserts the string "str" starting at position "pos" in the window.

In addition, the comtab which associates keys with functions is changed to "*hyper-comtab*" which contains the following associations:

- CONTROL-D: hyper-ed-delete-char.
- Backspace Key: hyper-ED-RUBOUT-CHAR
- Tab Key: hyper-ED-INDENT

The functions used are:

- hyper-ED-RUBOUT-CHAR (): A "hyper_fred_window" object function that is associated with the backspace key. It deletes the chaterbefore the current cursor mark.
- hyper-ED-INDENT (): A "hyper_fred_window" object function that is associated with the TAB key. It inserts four spaces properly by updating the positions of the affected links.
- hyper-ed-delete-char (): A "hyper_fred_window" object function that is associated with the CONTROL-D key combination. It deltes the character at the cursor's position and updates the positions of the affected links.

4.3 Node Addresses

Recall that node addresses are used as source and destination of links. The following functions are provided for node addresses:

- make_node_address(node_id start_pos end_pos): creates a new object of type node address, with the given information.
- copy_node_address (node_address): creates a new object which is a copy of the one given as as input.

4.4 Links

The set of active links is kept in memory during a session. At the end of a session it has to be placed on disk. A collection of links is called a "Web". The following set of functions is used to store the Webs in files. These functions are to be replaced by a call to a DBMS.

- reset_web (): erases the currently active set of links.
- set_web (web_fname): loads the web stored in the file named "web_fname".
- add_web (web_fname): adds the web stored in the file named "web_fname" to the current set of active links.
- save_web (web_fname): saves the current set of active links in file "web_fname".

Adding and removing liks from the current link repository (the current Web) is achieved via the functions:

- add_to_link_base (link), and
- delete_link (link).

The functions to manipulate links are defined in three files: create-link.lisp, link.lisp and buttons.lisp. Some of these functions are Lisp functions, others are object functions for different types of objects (not necessarily links).

Here is a list of the functions:

- create_new_link (source destination type &optional keyword_string): Checks if the link is not duplicating an existing one (with the function same_link_info). If not, it calls create_link to create the link.
- create_link (link_id source destination type &optional keyword_string): creates a new link and adds it to the list of links.
- select_source_of_link (): a "hyper_fred_window" object function. It is used to intertactivelly set the soure of a link when the user manually creates a link. It uses the global variable link_source.

- select_destination_of_link (): a "hyper_fred_window" object function. It is used to intertactivelly set the destination of a link when the user manually creates a link. It uses the global variable link_destination.
- cl (): a "hyper_fred_window" object function. It is used to intertactivelly create a link connecting the previously set source and destination.

The following are functions manipulate links:

- delete_all_links_in (list): deletes from "link_list" all the links appearing in list.
- delete_all_links_originating_in (node_name): deletes all links which originate in the node with name "node_name".
- delete_all_links_affected_by_selection (): a "hyper_fred_window" object function that will delete all links originating or ending in the region selected within the window.
- traverse (): a "link" object function, it accesses the destination node of the link, it also adds the link to the "backtrack" stack.
- traverse_link() : a "hyper_fred_window" object function that will identify the links affected by the current selection and traverse the first of these.
- get_links_with_source_affected() : a "hyper_fred_window" object function that will identify the links affected by the current selection.
- find_links_originating_here() : a "hyper_fred_window" object function that will identify the all links originating in a window.
- find_all_links_with_source_equal() : a function that will identify the all links originating in a node.
- collect_links_with_source (node_name list_of_links): extract from "list_of_links" all links that originate in node_name.
- find_links_ending_here() : a "hyper_fred_window" object function that will identify all links ending in a window.

- find_all_links_with_destination_equal (node_name): a function that will identify the all links ending in a node.
- collect_links_with_destination (node_name list_of_links): extract from "list_of_links" all links that end in node_name.
- highlight_links() : a "hyper_fred_window" object function that will highlight the all links originating in a window.
- highlight(): a "link" object function that will highlight the link in its originating window.
- un_highlight_links() : a "hyper_fred_window" object function that will un-highlight the all links originating in a window.
- font_spec: a "link" objevt function that will decide the font used to highlight a link depending on its type.
- move_links_over (position delta): a "hyper_fred_window" object function that update the position of all links either originating or ending in the winmdow from position given by the argument "position" by delta (positive or negative). This is used whenver inserting or deleting text.

Whenever a selection is active in a window, the set of links in that window is classified into seven sets accoring to the kind of overlap it has with the selecttion. The classes are:

- No match: links that do not intersect with the selection
- Partial Right: links that partially intersect on the rhs of the link.
- Partial Left: links that partially intersect on the lhs of the link.
- Partial Enclose: links that contain the selection.
- Exact: links that exactly match the selection.
- Enclosed: links that are enclosed by the selection
- Error: links that are not covered under the other alternatives, as programming mistake.

Last we have:

- links_clicked(): a "hyper_fred_window" object function that returns the six sets of links affected by the current selection (as described above).

4.4.1 Keyword links

Keywording is supported via the special link type "keword_link".

- create_keyword (keyword_string anchor type): makes "keyword_string" into a keyword. It associates its destination with "anchor", a node address. The type can be used to allowd further differentiation among different classes of keywords.
- current_keyword (): A "hyper_fred_window" object function that marks the text of the current selection as a keyword. Used to interactively define keywords.
- select_keyword_definition(): A "hyper_fred_window" object function that sets the definition of the current keyword to the selected text.
- create_links_for_keywords(): A "hyper_fred_window" object function that will identify all keywords occurring in the window.

5 System Details

The global variables used in the system are given next.

- link_source: the source of a link to be created. Used when manually creating links.
- link_destination: the destination of a link to be created. Used when manually creating links.
- clipboard_size: used to keep track of the size of the text being copied by the "copy" and "cut" functions. It is used to determine by the displacement of links when "paste" is performed.
- copy_links_list: used to keep a list of the links attached to the text being copied onto the clipboard by the "copy" "hyper_fred_window" object function.
- link_list: keeps the currently active list of links. (Loaded and saved onto a web)
- keywords_assoc_list: table associting a keyword string with a node adress (suposedly the location of its definition.
- current_keyword: the keyword being defined, as a string of characters.
- *hyper-comtab*: the association between keys and functions used for hypertext editing.
- active_node_list: The list of the currently active (open) nodes
- link_number: A counter. Each link is automatically given a number using this counter (which is incremented each time).
- hyper_emacs_nodes_menu: The menu for node operations.
- hyper_emacs_links_menu: The menu for link operations..
- hyper_emacs_webs_menu: The menu for web operations.

6 Code

The code is stored in the following files:

- start.lisp: start the system.
- hypersystem.lisp: define importany parameters, load the system.
- hypermacs.lisp: Nodes objects.
- links.lisp: Lisp objects.
- buttons.lisp: address objects.
- hypert-comtab.lisp: redefining the keyboard mapping.
- cut-paste.lisp: hypertext copy, cut & paste.
- menues.lisp : interactive menue definitions.
- link_data_base.lisp: storing the links.
- keywords.lisp: managing keywords.

6.1 start.lisp

Starts the system. It defines some parameters, sets the current directory as the root for all subsequent file management operations in the system. It the calls HYPERSYSTEM.LISP.

6.2 hypersystem.lisp

Declares theglobal variables.
Loades the rest of the system.

6.3 hypermacs.lisp

All node maintenance functions and Hyper-Windowing functions are defined here. It loads Hyper-Comtab.lisp.

6.4 Hyper-Comtab.lisp

Redefines the mapping between keyboard keystrokes and lisp functions. It customizes the keystrokes to function in the Hypert-Text Environment.

6.5 link.lisp

Defines Link Objects and their maintenance functions.

6.6 buttons.lisp

Defines node_adresses (my version of *buttons*) and all related functions such as:

- links originating in a node
- links ending in a node
- links affected by a selection
- etc.

6.7 cut-paste.lisp

Redefines the Cut, Copy & Paste operations to incorporate link manipulation.

6.8 menues.lisp

Defines all the menues to use when in manual-inteactive operation mode.

6.9 link_data_base.lisp

Defines all function related to Web manipulation. That is: loading and saving collections of links.

6.10 keywords.lisp

Defines keyword objects and their functions.