CharGer©

Conceptual Graph Software by Harry Delugach

User's Guide

Contents

Introduction 3
Features currently supported 3
File features 4
Editing features 4
Compatibility features 5
CGIF Compatibility 5
Features not currently supported 5
Why the name "CharGer"? 6
Installation 6
System Requirements 6
Software 6
Windows Install 6
macOS Install 6
Running .jar file alone 7
Files and folders used 7
Configuration File 8
Documentation 8
General Information 8
User Windows 8
The Graph Drawing Area 10
Main Window 10
Main Window Buttons 11
Main Window Menus 11
File Menu 11
Tools Menu 12
Windows Menu 12
Editing Window 12
Align Vertically 13
Align Horizontally 13
Menus 14
File Menu 14
Edit Menu 18
Draw Menu 21
View Menu 21
Operation Menu 22
Windows Menu 23
Drawing Tools 23
Selection Tool 23
Concept Tool 24
Relation Tool 24
Actor Tool 24

Link Tool 25
Coreferent (Line of Identity) Tool 25
Type Label Tool 25
Relation Type Label Tool 25
Generalization/Specialization Line Tool 25
Note Tool 25
Delete Tool 26
Shortcut Keys 26
Command Buttons 26
Make Context 26
Make Cut 27
UnMake Context 27
Preferences Window 27
Appearance 27
Current Default Colors 30
Compatibility 31
CGIF options 32
Other options 32
Actor Settings 33
Admin (use –diagnostics at startup) 34
CRAFT Settings (optional) 34
Actor Activation 35
Database Linking 38
CRAFT Subsystem 40
CRAFT main window 40
The Grid Window 41
External Module Plug in Interface 43
Known Bugs and Restrictions 43
Frequently Asked Questions 45
Technical Reference 46
***CharGer* XML File Format (version 3.1b and later) 46**
Repertory Grid Proprietary File Format (v. 3.2b and later) 51
***CharGer* Proprietary File Format (version 3.0b only) 52**
***CharGer* Proprietary File Format (versions 2.6b or earlier) 53**
Development Details 54
Invoking *CharGer* from an application 54
Actor Plugin Interface 54

Figures

Figure 1. CharGer Main Window. 9
Figure 2. Editing Window. 9
Figure 3. Basic Editing Features. 13
Figure 4. Formatting toolbar 13
Figure 5. English paraphrase window. 16

Figure 6. CGIF display window..... 17
Figure 7. Appearance Preferences panel..... 28
Figure 8. Enhanced "cut" display..... 29
Figure 9. Font Chooser Dialog..... 30
Figure 10. Compatibility panel. 31
Figure 11. CRAFT settings..... 35
Figure 12. Actor example. 36
Figure 13. Changed actor graph..... 36
Figure 14. Illustration of the <dbfind> and <lookup> actor. 38
Figure 15. Database Linking Tool Window. 39
Figure 16. CRAFT main window. 41
Figure 17. CRAFT repertory grid window. 42

Introduction

CharGer is a conceptual graph editor intended to support research projects and education. Its current version is primarily an editor to create visual display of graphs. It is a research tool for conceptual graph researchers to explore implementation issues in conceptual graph interfaces. For some specialized purposes, it also supports an interface to WordNet[1] (a popular dictionary/thesaurus), a repertory grid tool for acquiring requirements (CRAFT), and a generalized module “plug in” facility. For troubleshooting, see the section on Known Bugs and Limitations.

Users of the software should have some familiarity with conceptual graphs, including concepts and relations, type hierarchies and type/referent pairs. Knowing about actors will also be very helpful. For more information about conceptual graphs, see the Web page: http:;//conceptualgraphs.org.

If you use CharGer to prepare technical papers and presentations, you are requested to cite the following as a reference (in addition to this manual):

Harry S. Delugach, “Implementation and Visualization of Conceptual Graphs in Charger,” *Intl. Jour. of Conceptual Structures and Smart Applications*, vol. 2, no. 2, pp. 1-19, 2014.

Features currently supported

These features are currently supported:

[1] WordNet 3.0 Copyright 2006 by Princeton University. All rights reserved. THIS SOFTWARE AND DATABASE IS PROVIDED "AS IS" AND PRINCETON UNIVERSITY MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PRINCETON UNIVERSITY MAKES NO REPRESENTATIONS OR WARRANTIES OF MERCHANT- ABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE LICENSED SOFTWARE, DATABASE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS. The name of Princeton University or Princeton may not be used in advertising or publicity pertaining to distribution of the software and/or database. Title to copyright in this software, database and any associated documentation shall at all times remain with Princeton University and LICENSEE agrees to preserve same.

File features

- Save and retrieve graphs from files, in an XML format called CGX (see Appendix). These are ordinary XML text files and are portable across all platforms.
- Retrieve graphs from earlier CharGer files, in a proprietary (i.e., non-standard) text format. These graphs have the extension .cg and are portable across all platforms.
- Save graphs in Conceptual Graph Interchange Format (CGIF) standard interchange format conforming to ISO/IEC 24707:2007 Annex B. These graphs have the extension .cgif and are portable across all platforms. Currently under revision.
- Display a graph's CGIF version or a graph's (crude) natural language paraphrase, copy it to the clipboard as text, or save it to a text file.
- Save a graph in some graphics formats.

Editing features

- Copy/paste of graphs using an internal clipboard.
- Copy and paste from CharGer to other applications.
- Any number of graph windows may be opened for editing.
- Concepts, relations and actors are all supported for editing.
- Contexts and cuts are supported.
- Type and relation hierarchies are supported using a Charger-specific “generalization-specialization” link.
- Custom colors for each kind of graph object, and the ability to set one’s own default color scheme.
- Arbitrary text notes can be placed on a graph – they convey no semantics, but serve to annotate.
- Type and relation hierarchies may be edited and saved the same way as a graph, and may be intermixed on the same sheet of assertion.
- Contexts are supported, including arbitrary nesting. Negated contexts (“cuts”) are also supported.
- Save-before-close to prevent losing a graph, and other modifications to prevent losing data.
- Changes to graph elements are tracked with “history” records that are saved with the file.
- Zoom in and out on a graph’s view
- Several levels of undo/redo for editing changes (currently the last 15 changes)
- Adjustable text wrapping for object labels
- Ability to customize fonts for every node in a displayed graph
- Portability to all major platforms (i.e., as portable as Java based on JDK 1.7)
- Shortcuts to move and explicitly resize graph objects and contexts
- Shortcuts to switch editing tools.

Compatibility features

- Activation of some built-in actors, including several primitive ones for arithmetic and elementary operations, with an optional “animation” to show their operation.
- Database access through actors, although restricted to tab-separated text file “databases” at present.
- User-written (in Java) actor plug-ins with a published plug-in interface.
- Capability to automatically create a skeleton graph from a database suitable for providing database semantics.
- A natural language paraphrase feature, to paraphrase a graph in English (other languages on the way)
- Ability to set user preferences and save them between sessions.
- Some conceptual graph operations (e.g., join, match) (see Known Bugs and Restrictions)
- Ability to export / import CGIF (see below) including type hierarchies
- Ability to attach Wordnet glossary definitions or generic glossary entries to concepts and types
- Ability to perform some concept acquisition via repertory grids
- Ability to export a repertory grid to Burmeister (CXT) format.
- Ability to accommodate an external module (inserted into the Tools menu)

CGIF Compatibility

Charger’s support for CGIF has the following characteristics:

- Type identifiers and referent identifiers that contain only alphanumeric characters do not have to be delimited in any special way. For example, the identifier **Harry** may be rendered simply as **`Harry`** without delimiters. The set of alphanumeric characters is all the upper and lower case Roman letters, digits 0-9, and the underscore.
- All other type identifiers and referent identifiers must be delimited by double quotes; identical to the " symbol. A double quote itself may be included in the identifier as long as it is escaped as per Linux convention. For example, **Edwin ("Buzz") Aldrin** would be rendered in CGIF as **`"Edwin (\"Buzz\") Aldrin".`** The delimiting quotes are not part of the identifier as processed internally.
- These constraints are obeyed whenever CharGer is writing CGIF to a file.
- When reading from a CGIF file, the presence of any other quoting convention will results in unpredictable and probably undesirable behavior.

Features not currently supported

The following useful features are not currently supported; plans are to implement them in the future. Users are urged to note the list of limitations and bugs.

- Validation facilities (except for enforcing CG formation rules)
- Pasting (as in copy/paste) from other applications to CharGer windows.

- Ability for actors to be themselves defined in CG form (actor plug-ins must be written in Java,).

Why the name "CharGer"?

The University of Alabama in Huntsville has several sports teams nicknamed the "Chargers". A catchy name at that. Since the "CG" initials appear in it, it seemed a natural choice. And I like the notion of forging ahead, attacking research problems and leading the way. So there it is.

Installation

System Requirements

The minimum requirements for CharGer are:

- A color display, 800 x 600 resolution or higher.
- At least 1 GB of RAM

Software

There are three different ways to install CharGer on your machine. For Windows and macOS, there is an installer that will create a stand-alone application that you run just like any other. For Linux (or for Windows or macOS users who are already familiar with running .jar files under Java), the .jar file itself is also available. All these downloads are available at http://concept.cs.uah.edu/charger-download.jsp. Click on the option you want.

Windows Install

- Find the **`CharGer4-Installer.exe`** file; it should be in your default Downloads folder.
- Double-click it. You'll see a license agreement to accept, then click **Next**.
- After the installation is completed, CharGer will automatically run.
- The next time, you'll find CharGer in your list of available programs to run.

macOS Install

- Find "**`Charger4.dmg`**" under "**`Downloads`**". Double-click it and agree to the license terms. There should then be a disk in your Finder called " **`Charger4`**".
- Open the new disk named " **`Charger4`**". Drag "**`Charger4.app`**" to "Applications". If you've installed it before, you may see a warning that you're going to replace it with this version. If that appears, Click "Replace".
- Launch "**`Charger4.app`**" from your Applications folder. The first time, launch it *while holding down the "control" key,* then choose "Open" from the menu.

  NOTE: If you see a dialog warning you that CharGer4.app is from an unknown developer or that it was downloaded from the Internet, make sure you try double-clicking while holding the control key as above.

- Click "Open". After a moment, you should see the CharGer Main Window (see below).

Running .jar file alone

You must have Java installed (not Javascript) to run the software. A Java VM of version JDK 8 or higher is required (either a JRE or JDK); see http://java.oracle.com for information on how to get a freely available version, or use any commercial Java package. Java may already be installed on your system; to find out, get to a command prompt and enter "**`java –version`**". If the command doesn't return an error, make sure the version is 7 or higher.

To run the software, do either of the following:

1. Double-click the **`CharGer4.jar`** file that you downloaded, and you're on your way! Alternatively, you may want to run it from the command line as follows.
2. Run CharGer under JRE/JDK through the command-line interface, invoking the following command in the top-level folder:

   ...> **`java -jar CharGer.jar <arguments>`**

This should invoke the application and bring up the CharGer Main Window.

- If you are unable to run a **`jar`** file directly like this, you should be able to unpack the .jar file (using the "**`jar xvf CharGer.jar`**" command) and work with the classes themselves. In that case, the command to run CharGer would be as follows (note the "." for the class path):
- ...\CharGer> **`java –cp . charger.CharGer`**

COMMAND LINE ARGUMENTS

Charger accepts these command line arguments:

- **`-p <pathname>`** Tells Charger the folder to initially look for its graphs. Default is "**`Graphs`**" in the user's home folder.
- **`-stdOutputToLog (true|false)`** If true, then direct all standard output/error messages to the **`CharGerLog.txt`** inside the folder **`.edu.uah.Charger`** in your home folder. Default is **`true`**.
- **`-craft`** Tells Charger to enable the CRAFT Requirements subsystem. Default is absent.
- **`-diagnostics`** Tells Charger to show diagnostic information on the console. Enables Admin mode. Default is present.
- **`-"Module name"`** Tells Charger to invoke the "startup" method of an external module. There may be more than one such external module launched in this manner. No default.

Files and folders used

A simple folder structure is typical when running CharGer. Your own graphs and database folder can be located anywhere you want; see the Preferences Panel for how to tell CharGer where they are.

A brief explanation of these is as follows:

`Graphs` Where CharGer expects to find graphs (*.cgif, *.cgx files) by default. Graphs can be opened and saved from/to any folder, however. See the Compatibility Panel in Preferences.

`Databases` Where CharGer expects to find its tab-separated text databases for the **`<lookup>`** or **`<dbfind>`** actor. See below for more information. See the Actors Panel in Preferences.

`Grids` Where CharGer expects to find its RGX repertory grid files. If you are not using CharGer for repertory grid acquisition (i.e., if CRAFT is disabled), then you won't need this directory. See below for more information. See the Craft Panel in Preferences.

There is no particular reason why the Graphs, Databases and Grids directories are specified separately, except for convenience. There is a **`Set all folders`** button in Preferences that will allow you to select one common directory to hold all three kinds of files.

Configuration File

CharGer's preferences panel allows each user to save their own set of preferences. These are saved in the **`CharGerUserPrefs.props`** file within a "hidden" folder in your home folder named **`.edu.uah.Charger`**. There is only one configuration per user. Although users shouldn't usually need to edit this file directly (Save Preferences does this for you), the entries in the file consist of two kinds of lines:

- Parameter lines of the form expected by the Properties class:

  ```
  Parametername = parametervalue
  ```

- Comment lines are of the form

  ```
  # anything…..
  ```

Documentation

Documentation of CharGer consists of a set of HTML files (generated from javadoc) documenting the classes (primarily useful for developers), some tool tips and informative messages during execution, and this manual, which can be found at http://concept.cs.uah.edu/charger-download.jsp.

General Information

The CharGer software is offered without guarantees or warranty of any kind. Although it is unlikely to do serious damage to your computer files, it may not be useful at all. Please report all bugs, suggestions, etc. to the author: Harry Delugach (delugach@uah.edu). The usual disclaimers hold with respect to my responsibility for any problems you may have with this software. In other words, **use at your own risk**.

User Windows

There are several kinds of windows in CharGer. One is the Main Window, which looks something like Figure 1. The version number displayed on your Main Window may be different; yours reflects the actual version you're using.

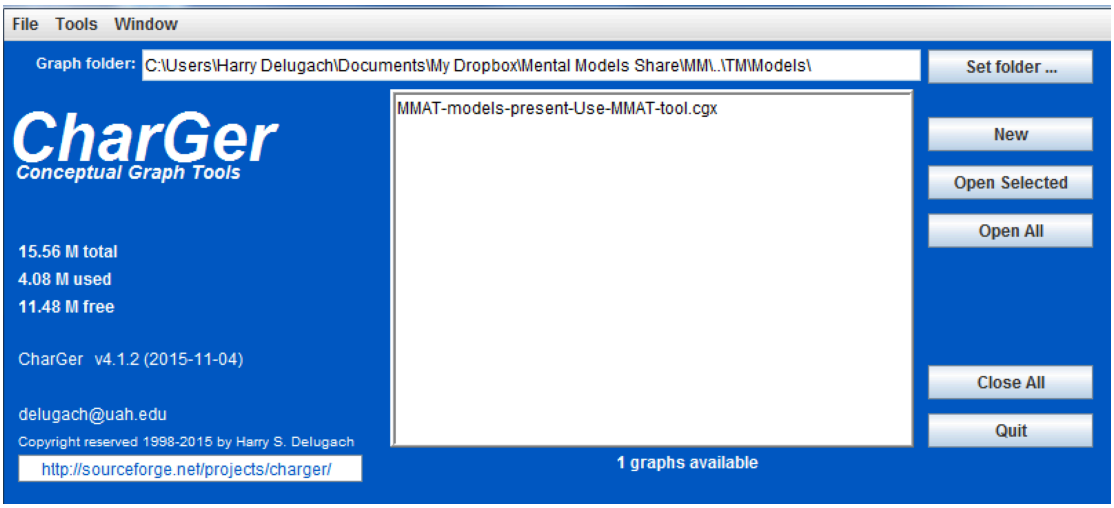


Figure 1. CharGer Main Window.

From this window, one or more editing windows may be opened. An editing window looks something like Figure 2. Details of its components and buttons are given below.

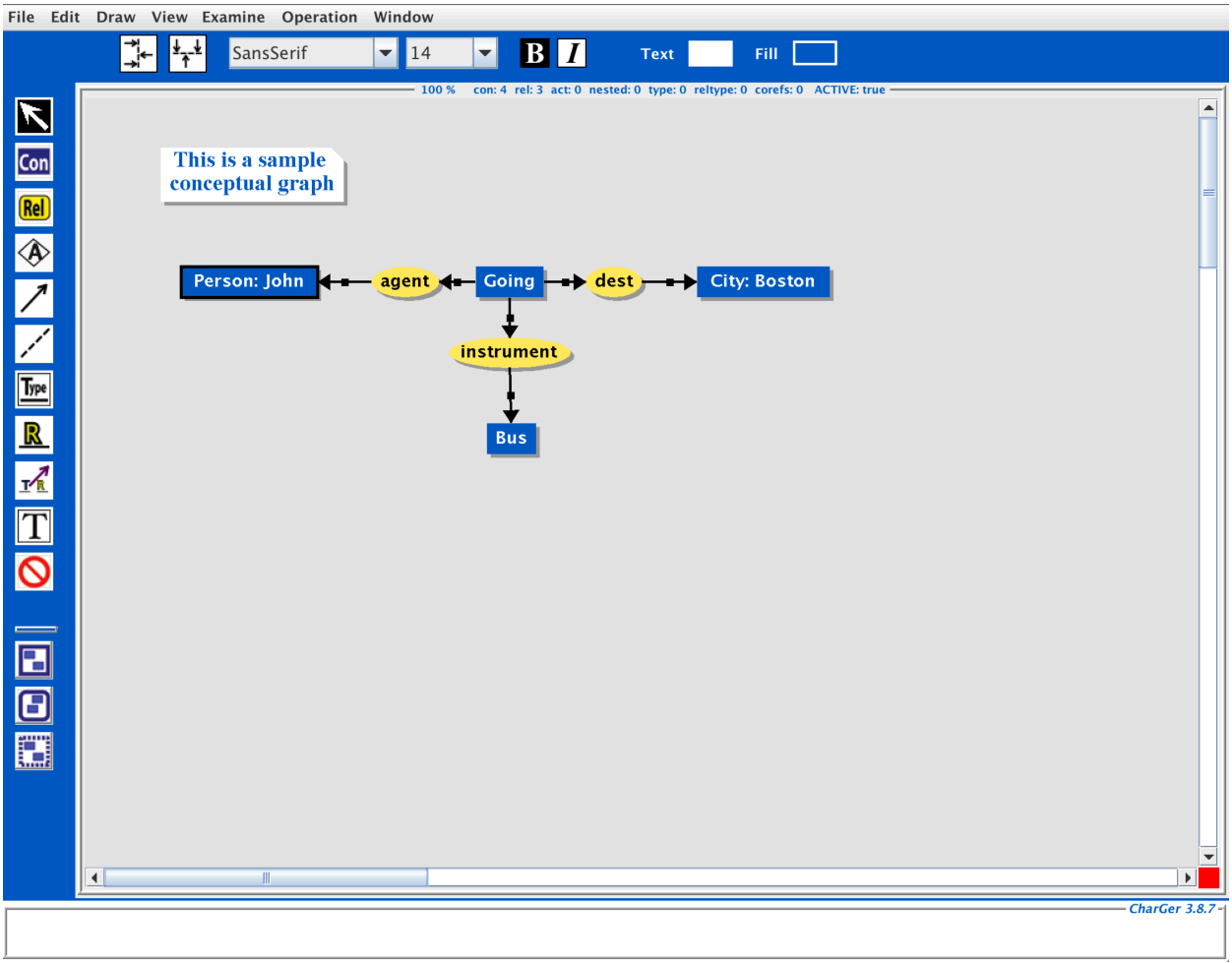


Figure 2. Editing Window.

There are additional window types (e.g., Database Linking Tool Window and other display windows) described below.

GETTING HELP

Many buttons, fields, etc. have "tool tips" that will help you understand what they do. To view a tool tip, place the cursor above some element in a window. After a short pause, a one-line help description will appear. If nothing appears, then there's no tool tip for that element; if you think it needs one, send a note to the author. In the meantime, look in this manual for more help.

GRAPH STRUCTURE

A **graph** in the CharGer environment is any collection of concepts, relations, actors and type labels, linked by their appropriate arrows, co-referent links, input/output arrows or super/sub-type arrows. A collection of un-connected elements is still considered a graph. All the elements of each graph are contained in a single file. The notion of multiple graphs means elements in multiple files.

We have introduced two new graphical elements not previously defined in conceptual graph display environments, namely, a **typelabel** and **relationlabel** shown as a text string above a horizontal line, as **ANIMAL**, This gives us an easy facility to draw type hierarchies and have them stored either separately, or included as part of another graph.

The Graph Drawing Area

There is a drawing area in an editing frame that serves as the "canvas" on which the user draws a graph. This area operates as a constrained version of a typical picture-drawing program, except that it will try to enforce the conceptual graph formation rules.

> **Note:** The "philosophy" behind the editor is that it supports an open world. That is, the user may enter anything he/she wants as long as it is well-formed by the general rules of conceptual graphs. If the user enters some element that is already defined, then the system will try to use that definition to enforce any constraints that result from it. If the user enters an element that is *not* already defined in the system, it can still be included, but no constraints will be enforced on it and little processing should be expected. Of course, the user is free to then provide definitions, facts, etc. that deal with the new elements.

Main Window

The main window lists the available graphs. This scrollable list has all the graphs available to the CharGer editor in the Graphs folder. The default naming scheme is as follows.

<GraphName>.cgx

Selecting one or more graphs from the list will cause each of them to be opened in an editing window when the **Open Selected** button is clicked. (see below).

> **Note**: Use the Quit button or menu item to end the session.

Graphs are currently saved in a standard XML form. This CharGer XML form is an easy-to-use text form, strictly for purposes of saving graphs within the CharGer editor. As text, they are intended to be portable across all platforms. This format is not intended as a replacement or alternative for the standard CGIF format.

Main Window Buttons

The following buttons in the Main Window are supported:

SET FOLDER…

Lets you select a directory where your graph files are located. Once chosen, it will become the default directory for opening/saving and all the .CG files in that directory will appear in the window.

NEW

Creates an empty editor window, ready for creating a graph from emptiness.

OPEN SELECTED

Opens the selected graph(s) in the displayed list. Double-clicking a single name will do the same thing for a single graph. This button only opens graphs from the list; to open graphs from anywhere else, use the Open… menu item in the File menu.

OPEN ALL

Opens the selected graph(s) in the displayed list. Double-clicking a single selection will do the same thing. This button opens all the graphs in the list from the default directory; to open graphs from anywhere else, use the Open… menu item in the File menu.

SAVE ALL CGIF

Converts every graph in the list to CGIF form.

CLOSE ALL

Close all editing windows. If a graph has been edited but not saved, you'll be prompted for each changed file as to whether you want to save it or not.

QUIT

Aborts the entire CharGer thread, after giving the user a chance to save any un-saved graphs or grids.

Main Window Menus

There are three menus associated with the main window — the **`File`** menu, the **`Tools`** menu, and **`Windows`** menu. The menus are explained below.

File Menu

NEW

Creates a new editing window. Does the same thing as the New button.

OPEN …

Lets the user choose a graph using a standard file selection dialog window. Operated the same way as the Open button.

OPEN CGIF …

Lets the user open a CGIF (*.cgif) graph as chosen through a standard file selection dialog window. CGIF is defined as per ISO/IEC 24707:2007 – Annex B.

PREFERENCES…

There is a preferences panel that allows the setting of some preferences by the user. Such settings are in effect for the current session only – CharGer always starts up with the preferences set from the **`Configuration`** file (see above).

QUIT

User gets a chance to save un-saved changes before finally quitting.

Tools Menu

There are three tools: the **`Database Linking Tool`** (described in Database Linking below) and **`Requirements Acquisition`** (see CRAFT subsystem, below). If any of these are dimmed, that means they are disabled, and you probably don't need to worry about it. Hint: if you want to enable all the tools, hold down the shift key and the control key while selecting "Tools" in the menu bar; this will enable all the tools for the remainder of your session.

(If you want to start up with the tools enabled, see the command line arguments for enabling them.)

Windows Menu

<GRAPH LIST>

If there are any already open graph editing windows, their names will appear here. The naming scheme is similar to the one that governs the file names (see above).

Editing Window

This is where most interesting work gets done. There are some editing and tool buttons to the left of the drawing area. The upper set of tool buttons are editing modes; the lower set are one-time command directives. Figure 3 is an overview of the primary editing window.

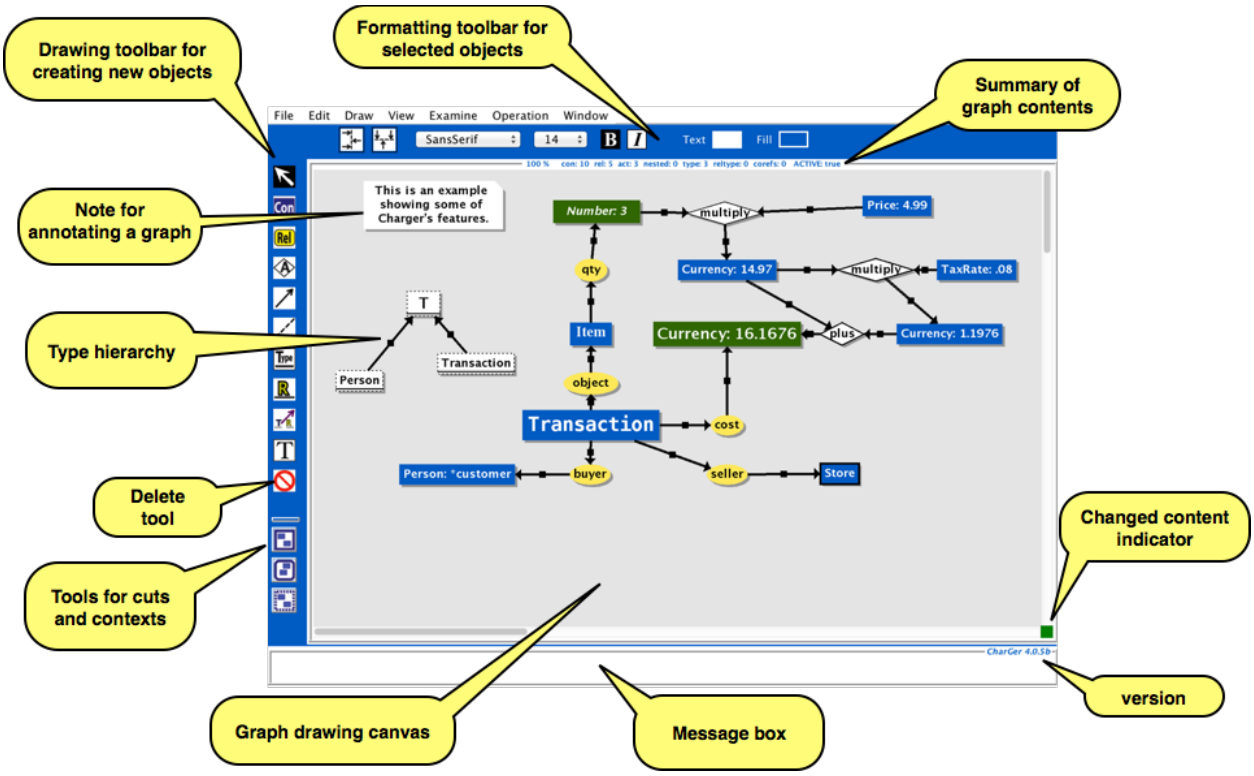


Figure 3. Basic Editing Features.

There are five menus on the Editing Window, three of which are shown in Menus below.

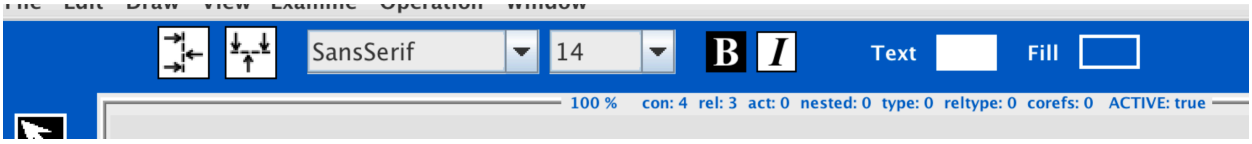


Figure 4. Formatting toolbar

The formatting toolbar is shown in Figure 4. Its features are similar to those available in the Edit Menu and in the Appearance preferences. There are buttons for aligning objects vertically and horizontally, pull-down menus for both font style and font name, as well as bold and italic button. Both the text color and fill color can be set for selected objects as well. When something is selected, the toolbar shows their current values. If the selected objects have different values, then nothing is shown, but any changes you make will be applied to all the objects that are selected.

Align Vertically

Aligns all objects in the current selection vertically, along their center points.

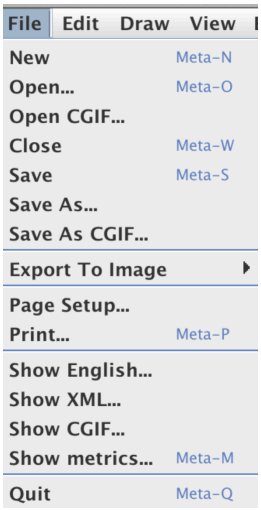
Align Horizontally

Aligns all objects in the current selection horizontally, along their center points.

Menus

Some of the menu items are also found as command buttons.

File Menu



The File Menu items are as follows. Many of them perform the same functions as the tools and commands below.

NEW

Opens a new editing window, with an empty graph.

OPEN …

Invokes a file open dialog for locating a graph file to be loaded. If the current file is un-saved, user has a chance to save it before opening a new one.

OPEN CGIF …

A parser for CGIF files is being developed. Some basic features of CGIF as per ISO/IEC 24707:2007 – Annex B have been implemented.

CLOSE

Closes the current graph. If it has not been saved in its present form, user is prompted before it closes.

SAVE

Saves the current graph without prompting for its name.

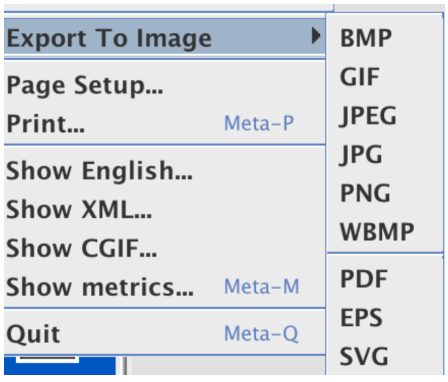
SAVE AS …

Invokes a file dialog for naming a graph file to be saved.

SAVE AS CGIF…

Opens a file dialog where the user can specify a file in which to save a CGIF version of the graph. The graph's CharGer information (i.e., the screen layout) will be embedded in the CGIF form if the "Export CharGer layout with CGIF" preference is checked in the Preferences Panel. Type hierarchy information will be included if “Export subtypes as relations in CGIF” is checked.

EXPORT TO IMAGE



Opens a sub-menu of format options that each open a file dialog where the user can specify a file in which to save an image of the file into several popular graphics formats. The file is then usable for inserting/editing by other software. The following export formats are generally supported:

Raster (bitmap, generally low resolution)

BMP (Windows Bitmap)

JPG, JPEG

GIF (Graphics Interchange Format)

PNG (Portable Network Graphics)

WBMP

Vector (line-oriented, generally higher resolution)

PDF (Portable Document Format)

EPS (Encapsulated Postcript)

SVG (Scalable Vector Graphics)

There is no way to import a graphics file into CharGer.

PAGE SETUP…

Invokes a page setup dialog, platform-specific to your operating system. Even if there are no changes to make, it's probably a good idea to click OK or Apply anyway (see Print… below). Page setup must be called once for each separate editor window you want to print.

PRINT…

Prints the display form of the graph on the currently selected printer. When invoking print in an editor window for the first time, you are first subjected to a page setup dialog with landscape mode already selected (see Page Setup above). It is recommended that you select landscape mode when printing out wide graphs. CharGer will attempt to scale your graph so that it will fit on a single page; there is no multi-page printing supported. (Page setup must be called once for each separate graph window you want to print.)

SHOW ENGLISH…

Create a natural language paraphrase of the current graph and displays it in a separate window, whose text can be selected and/or copied. The display looks like Displays the XML form of the graph in a separate window. The window's Edit menu allows you to copy part or all of the contents as text.

SHOW CGIF…

Displays the CGIF form of the graph in a separate window. The display looks like Figure 6. The window's Edit menu allows you to copy part or all of the contents.

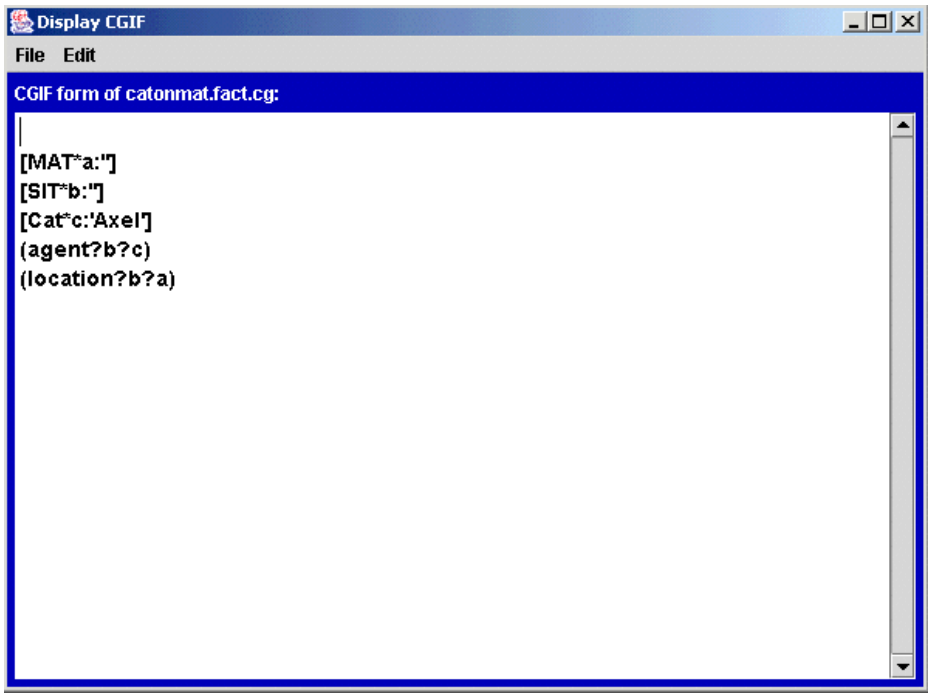


Figure 6. The natural language generating algorithm used here is a very simple one, primarily to illustrate feasibility. Only English is currently supported (apologies to my French-, German- and other-speaking friends).

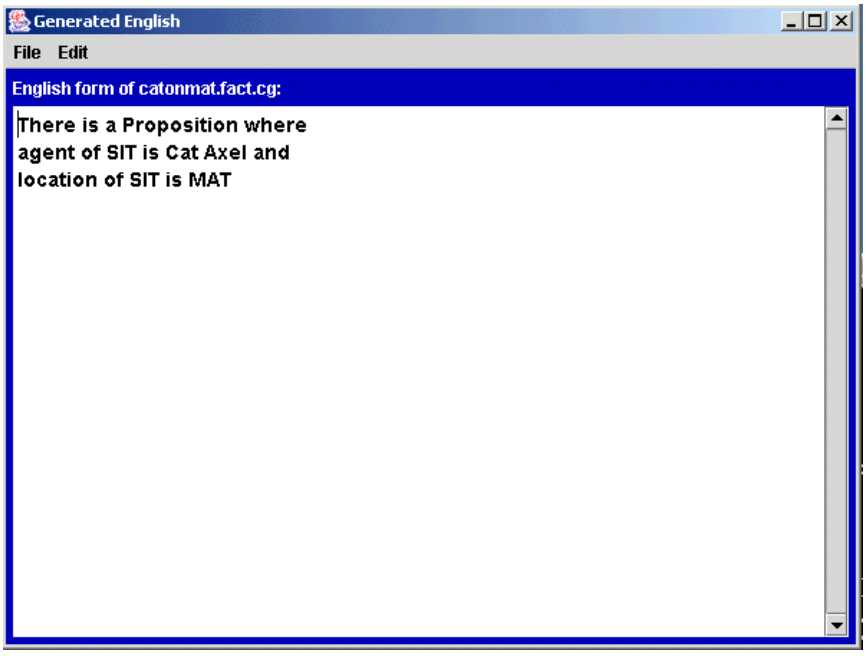


Figure 5. English paraphrase window.

SHOW XML…

Displays the XML form of the graph in a separate window. The window's Edit menu allows you to copy part or all of the contents as text.

SHOW CGIF…

Displays the CGIF form of the graph in a separate window. The display looks like Figure 6. The window's Edit menu allows you to copy part or all of the contents.

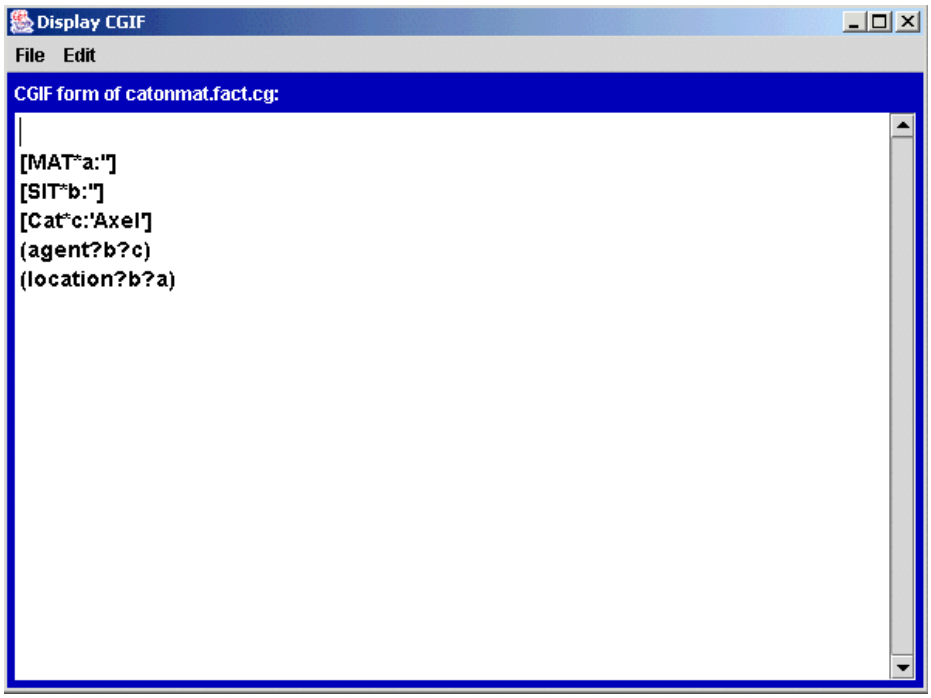


Figure 6. CGIF display window.

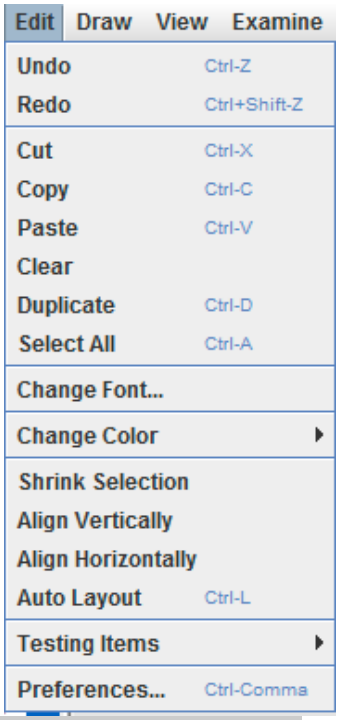
SHOW METRICS…

Some rudimentary metrics on the graph are displayed. This can be considered experimental at present; suggestions from users are welcome.

QUIT

User gets a chance to save un-saved changes before actually quitting.

Edit Menu



The items in the menu are as follows. Many of them perform identical functions to the tools and commands below.

Selections

When an edit item refers to “the selection” the reader should keep in mind some simple conventions:

- Edges (arrows, co-referent links, etc.) are in the selection if they are clicked on, shift-clicked on, or if the nodes they connect are selected.
- If a concept, relation, actor or type label is cut or cleared, its arrows, co-referent links, etc. are cleared with it, even if they lie outside the selection; it makes no sense to have an arrow without a graph node at both its ends.
- If a selection is pasted, only those arrows, co-referent links, etc. that are entirely within the selection are pasted; other arrows, co-referent links, etc. are not pasted, even if they were selected; this is because there is no practical way to connect the links’ other ends.

Note: Cut, Copy and Paste operate on either regular text (when editing labels in graphs) or on graph objects (when drawing). They operate separately, however; cutting or copying text has no effect on cut or copied graph objects and vice versa.

UNDO/REDO

CharGer supports a limited number of levels of "undo". Undo does not operate within the editing of a text field, although the previous contents of a text field (i.e., before you start editing) can be restored through Undo/Redo.

Undo restores the graph to its state prior to the last altering action. A subsequent undo restores the state prior to that one, and so on. Redo "undoes" undo -- i.e., it restores the graph to its state after the last altering action was performed. After choosing "undo" or "redo" any action other than a subsequent undo or redo will erase any further actions to "redo", but leave the "undo" actions unaffected. Dimmed if there is nothing prior to undo or nothing to redo.

CUT

Copies the selection to the (internal) clipboard and deletes it from the graph. **Undo** restores it, but leaves it on the clipboard. If the user is editing a text field (see below), the system clipboard is used, allowing transfer of the text to another application. If the user has selected one or more contexts, all the contents of the context are also selected and cut to the clipboard. Dimmed if there is no selection.

If the user then pastes into CharGer, they are pasted as CharGer objects and can be edited in the usual way. ~~If the user pastes into some other application, the selected objects will be pasted as a raster image and treated the same way that application treats other raster figures. The format of what’s pasted can be controlled by a preference in the Compatibility pane.~~

COPY

Copies the selection to the clipboard, leaving it intact for the current graph. If the user is editing a text field (see below), the system clipboard is used, allowing transfer of the text to another application. If the user has selected one or more contexts, all the contents of the context are also selected and copied to the clipboard. Dimmed if there is no selection.

If the user then pastes into CharGer, they are pasted as CharGer objects and can be edited in the usual way. ~~If the user pastes into some other application, the selected objects will be pasted as a raster image and treated the same way that application treats other raster figures. The format of what's pasted can be controlled by a preference in the Compatibility pane.~~

PASTE

Inserts the contents of the clipboard. If graph objects were cut/copied in CharGer, then they are pasted into the current editing window, where they can be edited in the usual way. **`Paste`** leaves the clipboard contents intact. If the user is editing a text field (see below), whatever text is on the system clipboard is used, even if it was copied/cut from another application. If the contents of the clipboard are from some other application besides CharGer, unpredictable actions may occur. Dimmed if there is nothing on the clipboard.

CLEAR

Erases all the selected objects, without affecting the clipboard. **`Undo`** restores them. Dimmed if there is no selection.

DUPLICATE

Makes a copy of the selected objects, without affecting the clipboard. They are automatically selected so that you can move the selection. **`Undo`** removes them. Dimmed if there is no selection.

SELECT ALL

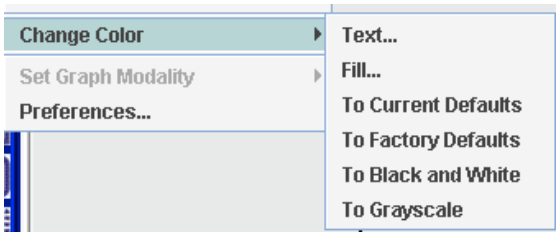
If editing in a text box, the entire contents of the text box are selected. Otherwise, everything on the drawing window is selected.

CHANGE FONT…

Select a new font for all selected items. Brings up a font chooser similar to the preferences one.

CHANGE COLOR

Change the color of the selected objects. Dimmed if there is no selection.



There are six choices:

Text… Change the text color of all selected objects, regardless of what kind.

Fill… Change the fill color of all selected objects, regardless of what kind.

To Current Defaults Change both the fill and text color of all selected objects to whatever default color scheme is currently set for their kind of object (see Preferences)

To Factory Defaults Change both the fill and text color of all selected objects to CharGer's "factory" defaults (you can't change these).

To Black and White Change the fill to white and the text color to black. Turn on the visible borders for each concept, relation, etc. Good for exporting and/or pasting when you're going to use a black and white printer.

To Grayscale Change the fill to a gray scale color and change the text to either black or white. Suitable for higher-resolution printing

SHRINK SELECTION

Make the selected objects as small as possible, shrinking around their centers. The same effect can be achieved by pressing the "-" key repeatedly. Dimmed if there is no selection.

ALIGN VERTICALLY

Aligns all objects in the current selection vertically, along their center points.

ALIGN HORIZONTALLY

Aligns all objects in the current selection horizontally, along their center points.

AUTO LAYOUT

Applies a force-based layout to the graph and re-arranges it so that it is visually more pleasing and attempting to eliminate overlapping objects. Can be repeated and/or undone to get more desirable results.

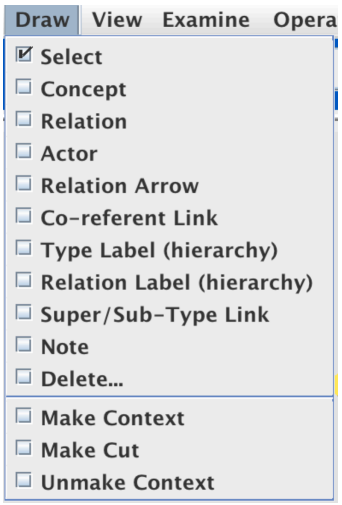
The algorithm attempts to make all edges the same preferred length. This can be changed in the Appearance Preferences (see below).

> **Note**: The default maximum number of iterations can be changed in the Admin panel (see below).

PREFERENCES…

Open the Preferences Window to make changes to the global configuration (see Configuration below).

Draw Menu



The Draw menu duplicates the commands available in the toolbar, for ease of accessibility. The tool modes of the editor are all available, as well as the following.

Make Context

Make the current selection into a context. Use the Selection Tool to establish a selection before using the MakeContext Tool. If the selection would cause overlapping contexts, the user is prevented with a warning. Dimmed if there is no selection, or if the selection is already a context/cut. There is no direct way to make a cut into a context – you must first unmake the cut and then re-make it into a context.

Make Cut

Make the current selection into a "cut" (negated context). Use the Selection Tool to establish a selection before using the **`MakeCut`** Tool. If the selection would cause overlapping contexts, the user is prevented with a warning. Dimmed if there is no selection or if the selection is already a context/cut. There is no direct way to make a context into a cut – you must first unmake the context and then re-make it into a cut.

UnMake Context/Cut

Removes the outermost context/cut border in the selection, thus attaching its contents to the graph in which it is nested (or the entire graph if the context was not nested). If there are concepts, etc. selected that are not in the outermost context, they are ignored. If there is more than one equally nested outermost context, one of them is chosen arbitrarily to be un-made. Dimmed if there is no selection.

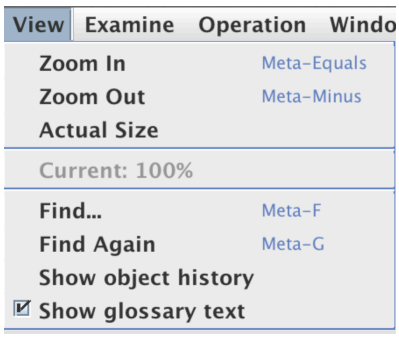
Align Horizontal

Same as Align Horizontal command button. Dimmed if there is no selection.

Align Vertical

Same as Align Vertical command button. Dimmed if there is no selection.

View Menu



The view menu aids in navigation and also controls some of the auxiliary content that is shown. Zooming is a feature of the graph's appearance on the screen at a given moment; the graph itself keeps the same size it had when you began zooming.

Zoom In

Increase the size of the graph's appearance by an increment (currently 10% each time).

ZOOM OUT

Decrease the size of the graph's appearance by an increment (currently 10% each time).

ACTUAL SIZE

Restore the graph's appearance to its normal 100% size.

CURRENT SIZE

Shows the current zooming percentage. This is also shown in the graph summary at the top border of the drawing canvas. Not selectable by the user.

FIND…

Allows searching for a string anywhere in the current graph. If not found, a message will appear in the message box.

FIND AGAIN

Look for the previous string again. If not found, a message will appear in the message box.

SHOW OBJECT HISTORY

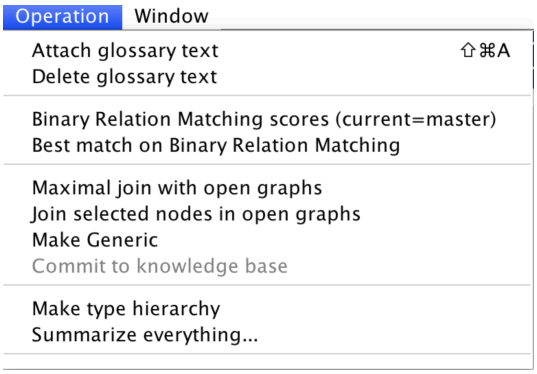
Shows the life story of this object within the current session. This will indicate whether it was inferred by an internal process, provided by the user, or read from a file.

SHOW GLOSSARY TEXT

Toggle to indicate whether glossary (definition) entries are to be displayed when some node is selected. This is either grayed out or absent unless Wordnet has been enabled in the configuration file.

Operation Menu

This menu invokes operations that can be performed on a graph or selection. In the future, there are more operations planned for this window.



ATTACH GLOSSARY TEXT

Allows the user to associate a glossary entry with this particular graph node. A glossary entry may be one of two kinds: one chosen from a set of Wordnet definitions or one supplied by the user. In order to use Wordnet definitions, you must have Wordnet 2.0 or later installed on your system.

DELETE GLOSSARY TEXT

Remove the associated text from the particular node. Of course, Wordnet itself is unaffected.

JOIN SELECTED NODES IN OPEN GRAPHS

Performs a join operation using two or more graphs, starting with the current graph. Each graph should have its own set of one or more selected items. If there is a valid way to join the graphs, according to the current matching scheme, a joined graph is created and opened in its own new window.

Note: Each graph has its own selection set, which can be maintained when switching between graphs. Be sure not to click on the drawing canvas when switching graphs, since clicking on the bare canvas will de-select whatever was previously selected. Use the graph window margin or the Windows menu.

MATCH TO OPEN GRAPHS

Matches the current graph to any other graphs in an open window. If a match is found, its result is displayed in a new window.

MAKE GENERIC

Remove the referents of the selected concepts and contexts. In other words, make individual concepts into generic ones. If a concept or graph has no referent, then it is unchanged. If a context has a graph descriptor, that descriptor is unchanged. Relations, actors, and types are unaffected.

MAKE TYPE HIERARCHY

Find all super- and sub-type links in all open files and attempt to construct a consistent hierarchy out of them. This hierarchy will be placed in a new untitled graph window and can be edited as the user wishes.

SUMMARIZE EVERYTHING

Using all open graphs and all open repertory grids (if any), paraphrase their content in poor English. The paraphrase appears in a text window that can be copied, saved, etc.

Windows Menu

The **`Windows`** menu consists of the following items:

<GRAPH LIST>

If there are any already open graph editing windows, their names will appear here. The naming scheme is similar to the one that governs the file names (see above).

BACK TO MAIN WINDOW

Allows convenient return to the main window.

Drawing Tools

Selection Tool

The selection tool is the basic way to make a selection or to move graph objects. In general, connecting lines are selected with what they connect. Double-clicking with the selection tool will invoke the EditText operation on whatever object has been selected.

- **To move a single object**, click on it and drag it where you want it. To move an object from one context to another, click on it and drag the same way.

Note: Moving an object in or out of a context will delete any links it has between relations and concepts. This is because CG rules don't allow a relation to

cross context boundaries. Coreferent links are preserved however. Links to actors can be preserved if the option is set in the Preferences Panel.

- **To select one or more objects as a group**, drag a rectangle across the objects you want selected. The objects should be "highlighted". You can then move them by moving any of the selected objects, or you can perform other operations. See **`Make Context`**, **`Shrink Selection`**, or the cut/copy/paste operations.
- **To add objects to a selection**, hold down the SHIFT key while dragging a new rectangle.
- **To select a context**, click somewhere on its outlined border.
- **To select an arc** (rendered as an arrow), a line of identity (coreferent link), or a generalization/specialization arrow, click on the solid dot (•) at its midpoint. In general, selection ignores arcs, because their meaning is inherent in the concepts/relations/actors/types that they link. In other words, as the nodes are selected/deleted/moved, so are their corresponding arcs.
- **To cancel a selection**, click somewhere on the drawing area that is not occupied by a graph element, or click another tool.

The selection tool can be chosen by pressing "S" or the space-bar.

Concept Tool

The concept tool allows you to insert a new concept onto the graph drawing area. To insert a concept on the drawing area, select the concept tool, then click where you want the new concept to appear. A new concept has a type label of **T** with no referent. To change a concept type or referent, open a Text Edit Box (see p. 25) on its name. The tool remains selected until another tool is selected. The concept tool can also be chosen by pressing the "C" key on the keyboard.

Relation Tool

The relation tool allows you to insert a new relation onto the graph drawing area. A new relation has the name **link**. To insert a relation on the graph drawing area, select the relation tool, then click where you want the new concept to appear. To change a relation's link name, open a Text Edit Box (see p. 25) on its name. The tool remains selected until another tool is selected. The relation tool can also be chosen by pressing the "R" key on the keyboard.

Actor Tool

The actor tool allows you to insert an actor onto the graph drawing area. To insert an actor on the graph drawing area, select the actor tool, then click where you want the new concept to appear. The actor has the name **`f`**, which can be changed with the text editing features. The tool remains selected until another tool is selected. The actor tool can also be chosen by pressing the "A" key on the keyboard.

Link Tool

The link tool allows you to connect concepts with relations or actors, according to the normal rules of conceptual graphs. To draw a link, select the tool, then click and drag from one concept (relation or actor) to a relation or actor (concept). The link tool can also be chosen by pressing the "." (period) key on the keyboard.

Coreferent (Line of Identity) Tool

This tool draws a coreferent link or line of identity between two concepts, thereby connecting members of a coreference set. To create a coreferent set of more than two members, each member must be linked (via the identity link tool) to at least one other member of the set. Coreferent links for a single coreference set should be drawn between a "dominant" concept and a "subordinate" concept. Redundant links are permitted but add no new information to the graph. The line of identity tool can also be chosen by pressing the "I" key on the keyboard (for line of identity)

Type Label Tool

The type label tool allows you to insert types (usually in a hierarchy) onto the graph drawing area. The type label tool can also be chosen by pressing the "T" key on the keyboard. The original formulation of conceptual graphs did not include any graphical way to show type labels, other than including them in concepts.

Relation Type Label Tool

The relation type label tool allows you to insert relation types (usually in a hierarchy) onto the graph drawing area. The relation type label tool can also be chosen by pressing the "L" key on the keyboard. The original formulation of conceptual graphs did not include any graphical way to show type labels, other than including them in relations.

Generalization/Specialization Line Tool

This tool draws a generalization/specialization relationship between two concept types or between two relations. The generalization/specialization tool can also be chosen by pressing the "," (comma) key (an un-shifted "<") on the keyboard. The original formulation of conceptual graphs did not include any graphical way to show subtypes or supertypes.

Note Tool

With this tool chosen, you can insert an arbitrary text note. This note appears as a “dog eared” box (similar to a UML note). It can be moved, edited, included in a context, etc. Although it does not affect the semantics (i.e., the meaning) of the graph, it can help explain something.

Delete Tool

With this tool chosen, clicking on a concept, relation, actor, context border, or type will delete it. To delete an arc (rendered as an arrow), click on the solid box (■) at its midpoint.

Shortcut Keys

Certain keystrokes can be used in the editing window, when not editing text. The shortcuts shown in menu items are available whenever their corresponding menu command is available. In addition, the following hotkeys perform some useful functions:

SHIFT adds to the current selection when clicking the mouse or dragging across a selection
increases increment when moving or resizing with arrow keys

ARROW KEYS:

LEFT moves the selected object(s) to the left one pixel (with SHIFT key, four pixels)

RIGHT moves the selected object(s) to the right one pixel (with SHIFT key, four pixels)

DOWN moves the selected object(s) down one pixel (with SHIFT key, four pixels)

UP moves the selected object(s) up one pixel (with SHIFT key, four pixels)

S or SPACE make the selection tool active

C make the concept tool active

A make the actor tool active

, (comma) make the arrow tool active (unshifted “>”)

I make the line of identity tool active

T make the type label tool active

L make the relation type label tool active

N make the note tool active

. (period) make the generalization/specialization tool (unshifted “<”)

= enlarge the selected objects by one pixel each (with SHIFT key, four pixels)

- (minus) reduce the selected objects by one pixel each (with SHIFT key, four pixels)
Note: due to a Java compatibility issue, you may need the "-" key on the numeric keypad to achieve this operation

Command Buttons

Make Context

Make the current selection into a context. Use the Selection Tool to establish a selection before using the **`MakeContext`** Tool. If the selection would cause overlapping contexts, the user is prevented with a warning.

Make Cut

Make the current selection into a negated context ("cut") displayed with a rounded rectangular border. Use the Selection Tool to establish a selection before using the **`MakeCut`** Tool. If the selection would cause overlapping contexts, the user is prevented with a warning.

UnMake Context

Removes the outermost context border in the selection, thus attaching its contents to the graph in which it is nested (or the entire graph if the context was not nested). If there are concepts, etc. selected that are not in the outermost context, they are ignored. If there is more than one equally nested outermost context, one of them is chosen arbitrarily to be un-made.

Preferences Window

There is a preferences window available that allows the user to adjust some settings for their own use. These settings will override those in the **`CharGerUserConfig.conf`** file, which is loaded at CharGer startup, but they will persist for that session only, unless the user presses the **`Save Preferences`** button. User-saved changes are in the optional **`CharGerUserConfig.conf`** file. No changes are made by CharGer to its own **`CharGerDefaultConfig.conf`** file; those changes must be made through a regular text editor outside of CharGer.

> **Note**: Changes in the **`Preferences`** window take effect immediately, except for font/color changes which are reflected in future windows, but not current ones.

Preferences are arranged in three panels, Appearance, Compatibility, and Actor Settings. All of them have the following common options.

Set all folders

Set the graphs folder, the database folder and the repertory grid folder (if CRAFT is enabled) to the same folder.

Make Preferences Permanent

Ordinarily, changes made in the preferences panel take effect immediately, but do not persist between sessions. In order to make the preferences permanent, choose Save Preferences. This updates the CharGerUserConfig.conf file (or creates it, if the file does not already exist).

Close

Dismiss the window.

Appearance

The appearance panel looks similar to Figure 7:

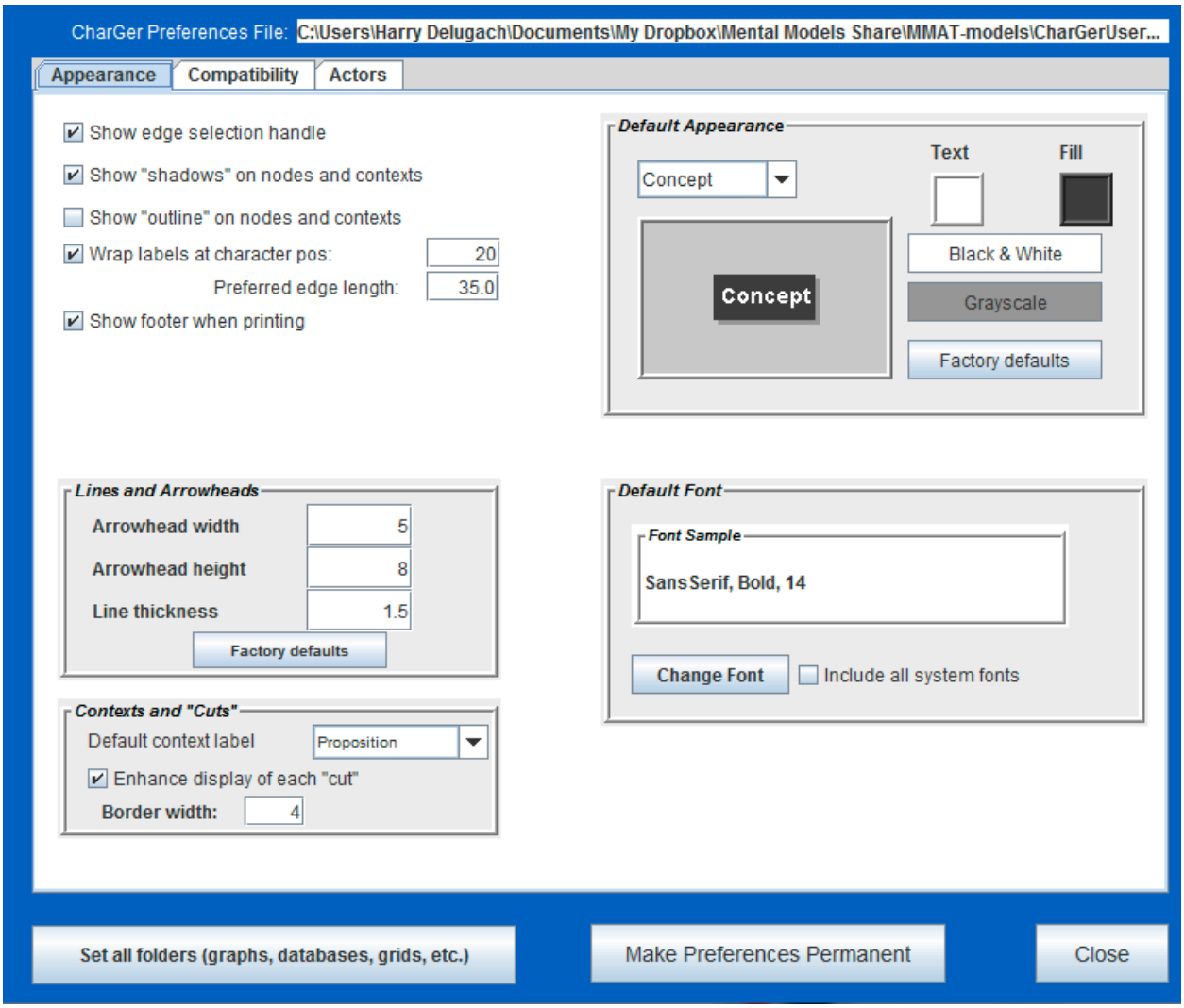


Figure 7. Appearance Preferences panel.

Show edge selection handle

Display the small selection box at the midpoint of each edge; this is automatically disabled for printing regardless of the setting shown.

Show "shadows" on nodes and contexts

Draw a gray "shadow" to give a three-dimensional effect. This option affects both the screen display and printing. The color sample will reflect the current setting.

Show “outline” on nodes and contexts

Show a solid border on all nodes and contexts; this is useful in black-and-white color schemes according to the user’s color preference. The color sample will reflect the current setting.

Wrap long node labels using wrap columns

Wrap text in labeled nodes so that they are a given number of columns wide (approximately).

Width (chars) to wrap

The approximate number of characters per line when wrapping text.

Preferred edge length

Used by Auto Layout to determine the best layout.

Show footer when printing

Include a footer when printing a graph that will show the graph's full path name.

LINES AND ARROWHEADS

Arrowhead width

How "spread out" to draw an arrowhead.

Arrowhead height

How "long" to draw an arrowhead

Line thickness

Thickness in pixels of lines, arrows and coreferent links.

CONTEXTS AND "CUTS"

Default context/cut label

Choose what label will be used for contexts or cuts when they are first formed. This was suggested as a way to make un-labeled cuts without having to edit the context name every time. Of course, the user can change any context's or cut's label by double-clicking on its border, which will bring up an editing box on the context/cut label.

Enhance display of each "cut"

Apply shading to the cuts so that they are more prominent and more distinct from non-negated requlars contexts. Figure 8 shows an example of the enhancement effect:

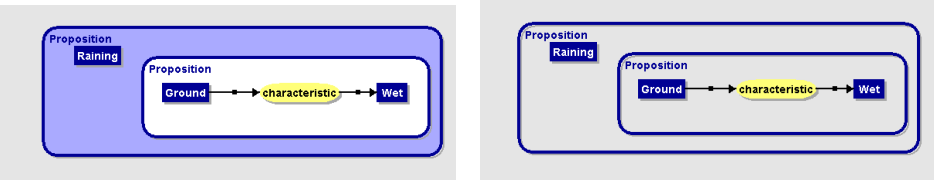


Figure 8. Enhanced "cut" display.

When checked When un-checked

Border Width

Width in pixels of a context's drawn border. Sets the margin width automatically.

Change Font

Brings up a brief dialog box to set the display font for graph labels. A sample showing the name, style and size of the font is shown. The dialog box looks something like Figure 9.

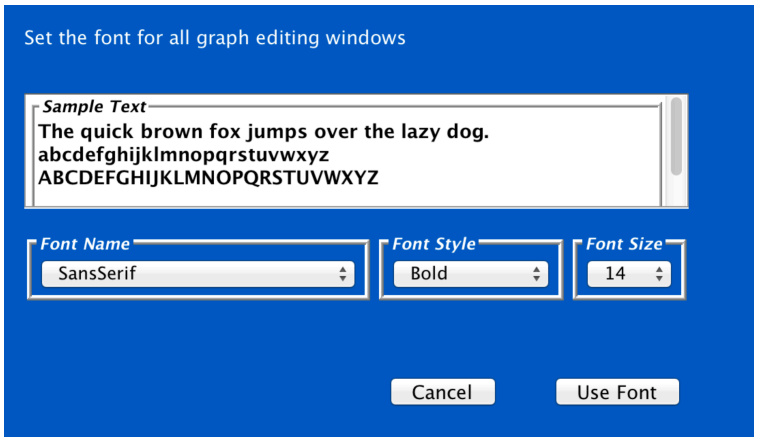


Figure 9. Font Chooser Dialog.

To make the displayed font active, click on "Use Font" in the dialog box, which will make the displayed font active and close the font window. This font change will take effect immediately but will not be remembered between sessions. To make the font change permanent for subsequent sessions, click on "Save Preferences" in the Preferences Panel.

Include all system fonts

Includes all fonts available on the system. This may be a very long list, and the selected font may not be available if the graph files are moved to another system where it may not be installed. If left un-checked, only a small set of platform-independent font names will be shown.

Current Default Colors

There are several features which you can use to control the colors of graphs you draw. They are grouped together in the Current Default Colors section of the Appearance Panel.

Object Pull-down menu (Concept, Relation, etc.)

Use this to select which kind of object's colors are to be examined/changed. The Color Sample area shows you what that object currently looks like.

Text

Click here to change the color of the text for the kind of object you've selected. The Color Sample area shows you what that object currently looks like. No graphs are changed yet.

Fill

Click here to change the color of the fill for the kind of object you've selected. The Color Sample area shows you what that object currently looks like. No graphs are changed yet.

Factory Defaults

Changes the text and fill colors to the factory default scheme (you cannot change it). No current graphs are changed; that is accomplished by the Change Color menu item in the editing window. The Color Sample area shows you what the current object looks like; however, ALL object types have been restored to these defaults.

Grayscale

Changes the text and fill colors to a pre-defined grayscale scheme (you cannot change it). This scheme can be useful when printing to a black and white printer. No current graphs are changed;

that is accomplished by the Change Color menu item in the editing window. The Color Sample area shows you what the current object looks like; however, ALL object types have been restored to these defaults.

Black and White

Changes the text and fill colors to a pre-defined black and white scheme (you cannot change it). This scheme can be useful when printing to a black and white printer. No current graphs are changed; that is accomplished by the Change Color menu item in the editing window. The Color Sample area shows you what the current object looks like; however, ALL object types have been restored to these defaults.

Make these preferences permanent

See above.

Compatibility

The Compatibility panel looks like this:

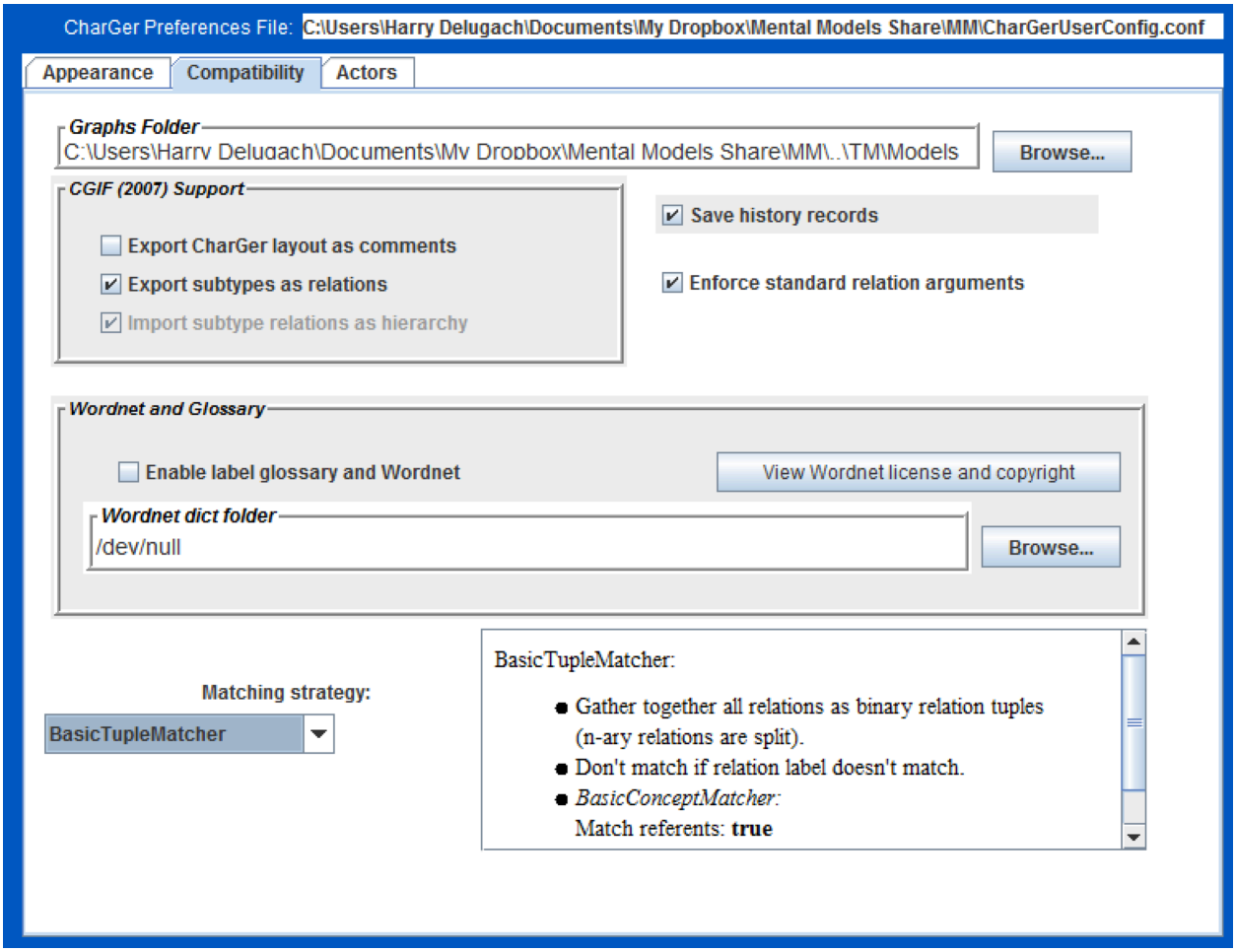


Figure 10. Compatibility panel.

Graphs Folder

The default folder from which the graph list in the main window will be constructed. Saved graphs will ordinarily be saved in the same directory from which they were read. In open and save file

dialogs, the user can navigate to whatever folder they wish. The **Set folder…** button allows the user to change this default folder. **Set all folders…** sets the graph, database and grid folders to the same name.

CGIF options

Export CharGer layout as comments

Include the CharGer layout information as CGIF comments, most importantly each component's layout on the canvas. This affects both the Display CGIF command and saving to a CGIF file. The information kept is the same information that appears in the <layout> xml tag of a .cgx file. The entire tag itself is wrapped in a CGIF comment that appears immediately after the opening "[" or "(" of a CGIF concept or relation. The comment is of the following form:

```
/**CG4L; <layout> ….. </layout> */
```

The introductory string **"*CG4L;"** can be changed in the Charger source through the Global.CharGerCGIFCommentStart variable.

Export subtypes as relations

This option controls whether to export subtypes or not. Since subtypes are not explicitly defined in Common Logic, CGIF does not have an explicit way to represent subtypes. The convention adopted in Charger is the following: Given a type label A and another type label B, Charger will generate the following to indicate that A is a subtype of B:

```
[Type: A] [Type: B] (subtype A B)
```

As a matter of convention, Charger will write these before any other type or relation elements within the same graph.

Other options

Save history records

Save the record of changes to this graph along with the graph itself. Note that the changes are recorded internally in all cases, and any history records in a graph file are automatically read in. This option affects only the storing of such records. **Note: If un-checked, then ALL history is removed when the graph is saved, not just the current session.**

Enforce standard relation arguments

Enforce the standard relation rule such that a relation has at most one output argument. Never enforced for actors, except that pre-defined actors must adhere to their specified input/output arity signatures.

Enable label glossary and Wordnet

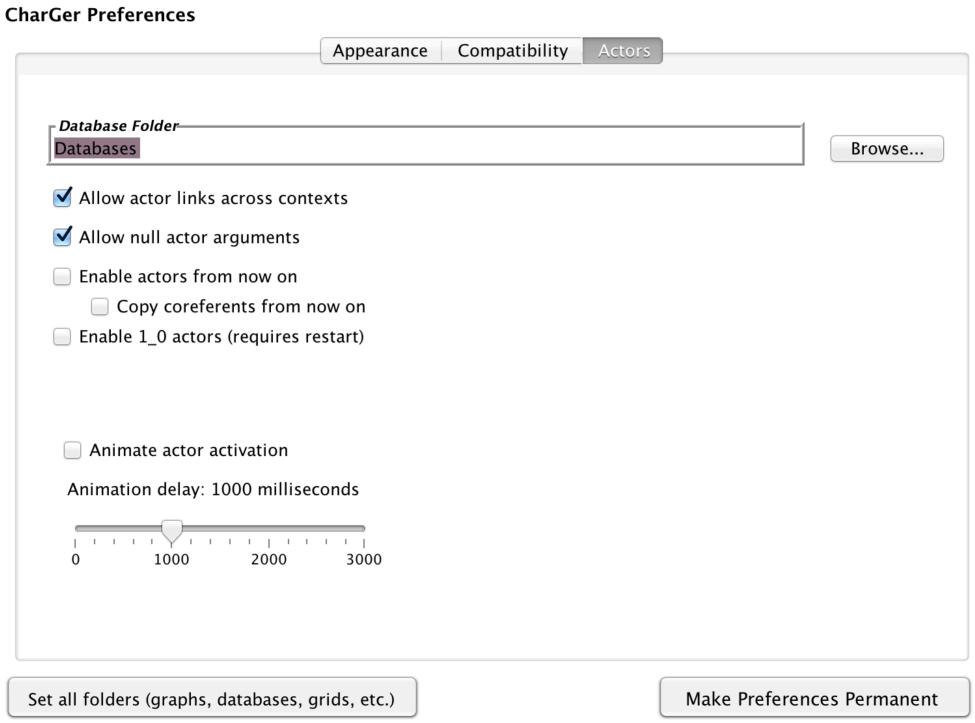
Enable the feature of including a glossary entry with each concept or relation. If Wordnet is not available, you can still include your own glossary entries with each concept or relation.

Wordnet dict folder

Wordnet's dictionary is a set of files usually found within a folder named dict; here is where you specify its actual location. Browse… lets you look for the folder's files on your system. This should usually need to be set only once; moving Wordnet around on your system ought to be a rare occurrence.

Actor Settings

The Actor Settings panel looks like this:



Database Folder

The default folder in which database files will be looked for. Database files are used in activating actors (see **Database Linking Tool**). The **Set folder…** button allows the user to change this default folder. **Set all folders…** sets the graph, database and grid folders to the same name.

Allow actor links across contexts

The ANSI standard does not allow relation links across context boundaries (whether an actor or relation). Strictly speaking, an actor should link to a concept in its own context, with a coreferent link from that concept into the other context. Checking this box allows an actor link to connect to any concept, regardless of its context.

Allow null actor arguments

If checked, considers a null argument legal to an actor. Usually means that the actor will ignore the argument.

Enable actors and copy referents from now on

If checked, actors will be enabled for the rest of the session. Un-checking it later may cause unpredictable and/or undesirable consequences, particularly if some actors involve executable plug-ins that are active.

Animate actor activation

Whether to activate actor firings in increments, visually marking those actors and concepts that are involved in each firing.

Animation delay

The time increment between firings, in milliseconds. Not visible unless actor firing is animated.

> **Note:** The Preferences panel is under constant development. Use tool tips to find out more about individual options – hold the cursor over the option you want to find out more about.
>
> **Warning**: If an option is shown in the preferences panel as "unstable", use them at your own risk; they are unlikely to help you out very much and will likely lead to errors.

Enable 1_0 actors (requires save and restart)

There are a number of built-in comparison actors that produce a zero or one, rather than a true or false. If you want to use these, then check this box, save the preferences and then quit and restart CharGer to take effect.

Admin (use –diagnostics at startup)

Although most users will be able to ignore this panel even if it present, it is documented here. To enter admin mode, use –diagnostics as one of the options on the command line.

Show internal info (boring)

Display some internal information in the editing window, and on the console (command-prompt window). This is primarily useful in reporting a problem to the developer(s). Most users probably want to leave it unchecked.

Number of iterations (Spring Layout Algorithm)

Change the default number of times the spring layout operates on the graph. The default is currently 5000. If the algorithm detects a nearly optimal layout, it will stop before that. Note that a user may invoke the auto layout option as many times as they want until they're satisfied with the appearance.

CRAFT Settings (optional)

> **Note:** If you aren't interested in the repertory grid tools, then ignore this section! Also make sure that craftEnabled is set to false in your local configuration file.

CharGer has a built-in experimental repertory grid interface that can be used for your entertainment. To enable CRAFT, quit CharGer and use the "-craft" command line option when running it.

These settings involve the CRAFT knowledge acquisition subsystem. The CRAFT Settings panel looks like this:

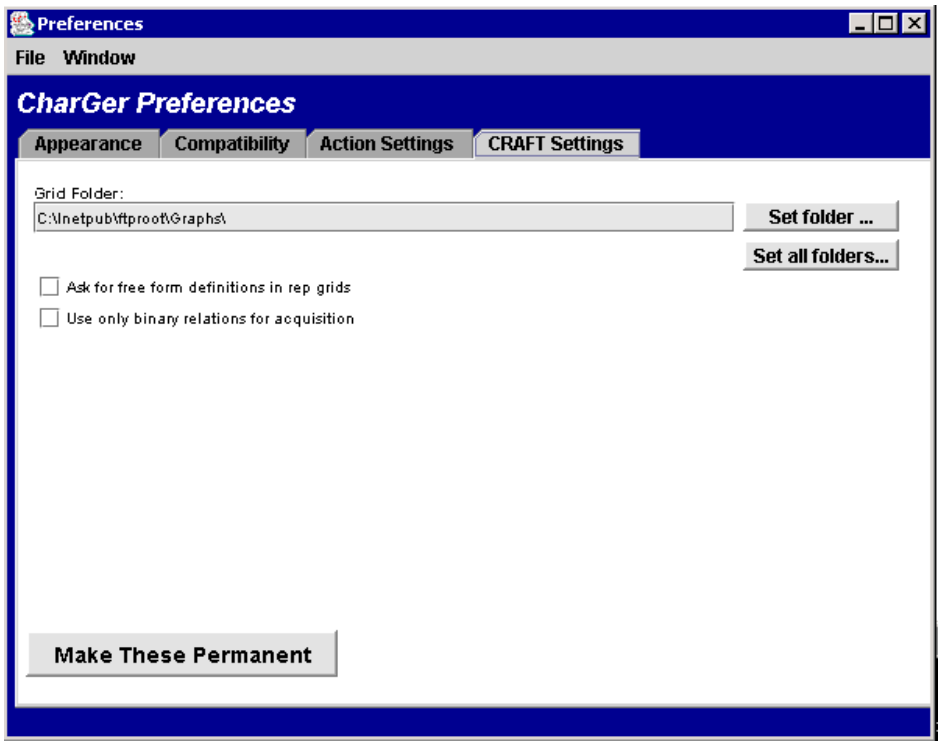


Figure 11. CRAFT settings

Grid Folder

The default folder in which repertory grid files will be looked for. Grid files are the result of acquisition using a repertory grid (see **Requirements Acquisition**). The **Set folder…** button allows the user to change this default folder. **Set all folders…** sets the graph, database and grid folders to the same name.

Ask for free form definitions in rep grids

If checked, will allow the user to provide their own definitions in addition to Wordnet's. If un-checked, then only Wordnet senses will be accessed. If Wordnet is unavailable and this is un-checked, then no glossary entries (definitions) will be accessible.

Use only binary relations for acquisition

If checked, only relations with exactly one input and one output will be included in the CRAFT main window. If un-checked, every pair of related concepts (or contexts) will be included in the CRAFT main window.

Actor Activation

CharGer supports actors, generally using the techniques described in Sowa's original book. There are a few built-in actors, with pre-defined semantics. Most of these are simple arithmetic operations. For example, there is a plus actor to implement addition. Consider the following graph. It would be good practice for you to draw the graph in Figure 12 before proceeding:

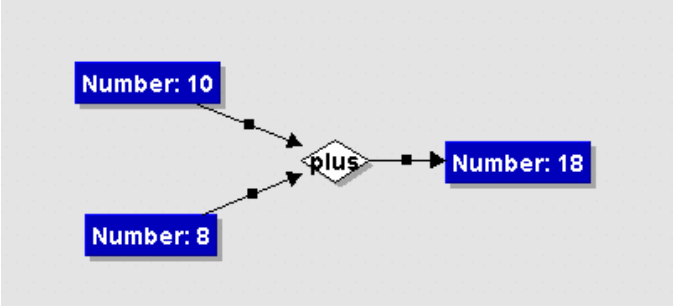


Figure 12. Actor example.

Note that the output number's referent is the sum of the two input number's referents. To understand how an actor works, draw the graph above, and then change the input **`Number: 10`** to read **`Number: 5`**. Note how the output number changes, as shown in Figure 13:

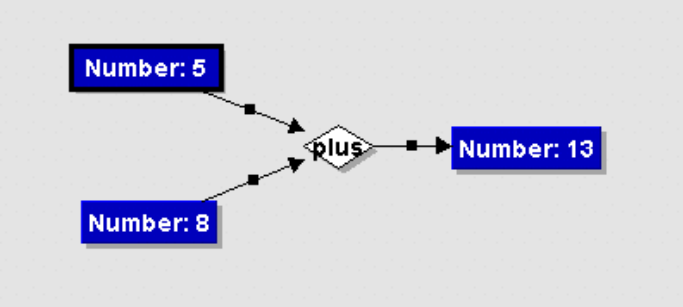


Figure 13. Changed actor graph.

Now change the output concept **`Number: 13`** to some other number. Note how it changes back to 13. The reason is that the plus actor denotes a functional dependency where the output concept is functionally dependent upon the input concepts; thus changing it causes the graph to re-evaluate itself and restore the original constraint.

The following executable actors are built-in to CharGer. ⊤ means any type; null means bottom (⊥). In general, actors' inputs are not commutative (i.e., their order matters, as denoted by the numbers on their input arcs.) Future versions will also allow actors to have a varying number of input concepts, when the meaning would be clear (e.g., **`plus`** could have two or more numbers to be added).

Compatibility Note: The actors whose names contain "1_0" are numeric versions of the logical actors returning T or null. They are by default disabled. You can enable them using the Preferences Panel.

	Input Concepts		Output Concepts		
Actor Name	Number	Type(s)	Number	Type(s)	Semantics
`copy`	One	**`T`**	One	**`T`**	Input (referent only) is copied to output concept.

dbfind **lookup**	Two	**Database** **T**	One	**Number**	Output concept is the value associated with **T**'s referent in the file denoted by the **Database** concept.
divide	Two	**Number**	One	**Number**	Output number is concept 1 divided by concept 2.
displaybar	One	**Number**	None		Input number is displayed as a bar in a separate window.
equal	Two	**T**	One	**T or null**	Output type is **T** if inputs are strictly equal; otherwise **null**
equal_1_0	Two	**T**	One	**1 or 0**	Output referent is 1 if inputs are strictly equal; otherwise 0
exp	One	**Number**	One	**Number**	Output number is input's referent raised to the power 2.718….
greaterequal	Two	**T**	One	**T or null**	Output type is **T** if input 1 greater than or equal to input 2; otherwise **null**
greaterequal_1_0	Two	**T**	One	**1 or 0**	Output referent is 1 if input 1 greater than or equal to input 2; otherwise 0
greaterthan	Two	**T**	One	**T or null**	Output type is **T** if input 1 greater than input 2; otherwise **null**
greaterthan_1_0	Two	**T**	One	**1 or 0**	Output referent is 1 if input 1 greater than input 2; otherwise 0
lessequal	Two	**T**	One	**T or null**	Output type is **T** if input 1 less than or equal to input 2; otherwise **null**
lessequal_1_0	Two	**T**	One	**1 or 0**	Output referent is 1 if input 1 less than or equal to input 2; otherwise 0
lessthan	Two	**T**	One	**T or null**	Output type is **T** if input 1 less than input 2; otherwise **null**
lessthan_1_0	Two	**T**	One	**1 or 0**	Output referent is 1 if input 1 less than input 2; otherwise 0
minus	Two	**Number**	One	**Number**	Output number is concept 1 minus concept 2.
multiply	Two	**Number**	One	**Number**	Output number is product of the two inputs. Commutative.
notequal	Two	**T**	One	**T or null**	Output type is **T** if inputs are strictly not equal; otherwise **null**
notequal_1_0	Two	**T**	One	**1 or 0**	Output referent is 1 if inputs are strictly not equal; otherwise 0
plus	Two	**Number**	One	**Number**	Output number is sum of the two inputs. Commutative.

There is as yet no actor-definition mechanism in CGs; that is a future enhancement. (Suggestions for appropriate mechanisms are welcome.)

An actor plug-in interface has been developed as of CharGer 2.6b or later. See the technical reference below for details on how to write your own actors for CharGer.

Database Linking

One of CharGer's features is its ability to use an external "database" that actor <lookup> can use. This ability is built into CharGer's actor definitions. For CharGer's purposes (as of the current version) a database file is a tab-separated tabular text file. For example, suppose the file DBElement.txt contains the following tab-separated values:

Number	Element	Symbol
1	Hydrogen	H
2	Helium	He
3	Lithium	Li
…	…	…

CharGer's **<lookup>** actor is designed to illustrate CharGer's interface to a database. The graph in Figure 14 shows how the database would work.

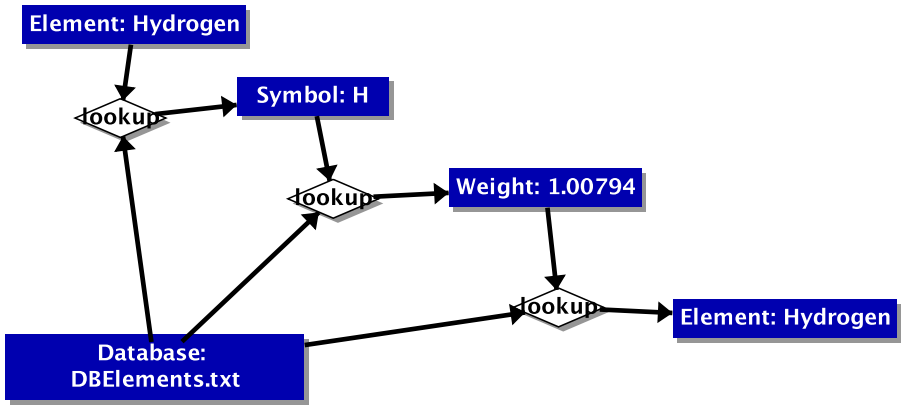


Figure 14. Illustration of the <dbfind> and <lookup> actor.

To see how the actor works, use the "T" tool to change "Hydrogen" to "Lithium". Note how the Symbol and Number concepts now have new referents! Another example is to change the Element referent to "Earth" (which is not an element) and note how both Symbol and Number become null (which is equivalent to ⊥ in conceptual graph terms).

There are a few important guidelines for how the **lookup** actor works. First, it requires an input concept of type **Database** whose referent is a real file name. Second, another input must have a type which exactly matches some field type (i.e., a header name from the file). Third, there must be a single output concept, whose type also matches some field type. (It's permitted for the input and output types to be the same; it's a good way to see whether the input concept's referent value is actually found in the database.) Use the **Database Linking Tool** to see what type names are valid in a given database.

A database with no tab characters will be treated as a one-column table whose entire first line is considered the only valid field name.

Restrictions: At present all databases must be in the same folder; the default is the **`Databases`** folder under the "top-level" folder; but this may be changed in the Preferences panel. Other restrictions probably exist regarding spaces in field names and things like that. There is also no type checking performed with actual values from the database; i.e., if the value of field **`Number`** is not a number, **`lookup`** won't care.

To make it easier for users to create usable graphs with database actors, the **`Database Linking Tool`** window has been provided. It is accessed through the **`Tool`** menu in the Main Window. When activated, the window looks something like Figure 15:

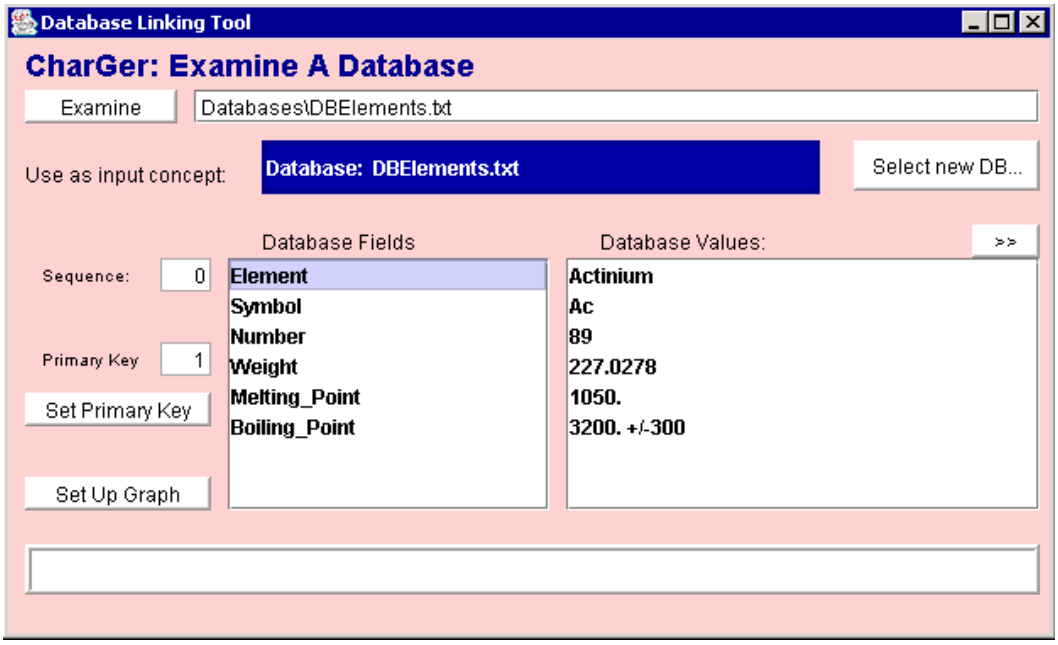


Figure 15. Database Linking Tool Window.

This window helps the user determine what are valid inputs and outputs to the database lookup actor(s). The window can also be used to set up a template graph for defining a database's semantics (with or without a primary key).

Examine: will open the "database" file whose name is shown. The first line of the file must be a header consisting of a series of two or more tab-separated strings, each of which is the field name corresponding to the subsequent lines in the file. At present, CharGer only works with text-only, tab-separated lines.

Select new DB… will open a file dialog giving the user a chance to pick a database file. The database file must reside in the chosen Databases folder; CharGer's default, or a folder selected by the user. This is a known limitation to be rectified in the future.

Note: Regardless of the folder one has reached through the file dialog, it will be ignored and the folder "Database" substituted.

Use as input concept indicates the valid input concept representing the database, to be used as an input to a <lookup> actor for effective lookup operations using this database.

Database Fields contains the exact names of the columns in the database file. These names are to be used as type-labels for the input concepts to the lookup actor. Referent values for such actors can be any value for the type. For example, [Number: 5] would be a valid input concept for the

fields given in the example screen. Selecting one of the database fields puts its sequence number into the **Sequence** box.

Database Values contains the values, for a given record (line) in the database file. The "**>>**" button skips to the next line. This lets the user confirm that the database is well formed and that the field names correspond to the desired values. There is no way to skip backward.

Set Primary Key makes the selected field name into the database file's primary key. When one of the field names is selected, show index will give its sequence number in the list. This is the same sequence number that will be used for the primary key sequence number.

Set Up Graph creates a new graph, in an editing window, with the database concept as input to a set of <lookup> actors, a primary key set up as the other input (if a primary key has been selected), so that the user can begin a graph already connected to a database. This is a handy tool for describing database semantics in a conceptual graph, and checking the semantics of the graph using actual values in a database.

CRAFT Subsystem

As of CharGer 3.3b, CharGer has been extended to include a repertory grid acquisition interface called the Conceptual Requirements Acquisition and Formation Tool (CRAFT). This tool allows repertory grids to be acquired based on an underlying conceptual graph. The workflow supported is as follows:

- ☐ Create one or more conceptual graphs with generic concepts representing parts of the domain you want to model.
- ☐ Make sure the graphs you want are opened in CharGer.
- ☐ If you want to include a type hierarchy, have exactly one CharGer window with type labels in it. These can be included with other graphs if you want.
- ☐ Invoke the CRAFT main window. It should display a list of phrases with their associated concepts and relationships.
- ☐ Choose a line and invoke "Start Grid". A new blank grid will open and a simple acquisition dialog will proceed.

There are two relevant windows and a glossary dialog box associated with CRAFT. They are explained in this section.

CRAFT main window

The CRAFT main window looks like **Error! Reference source not found.**Figure 16:

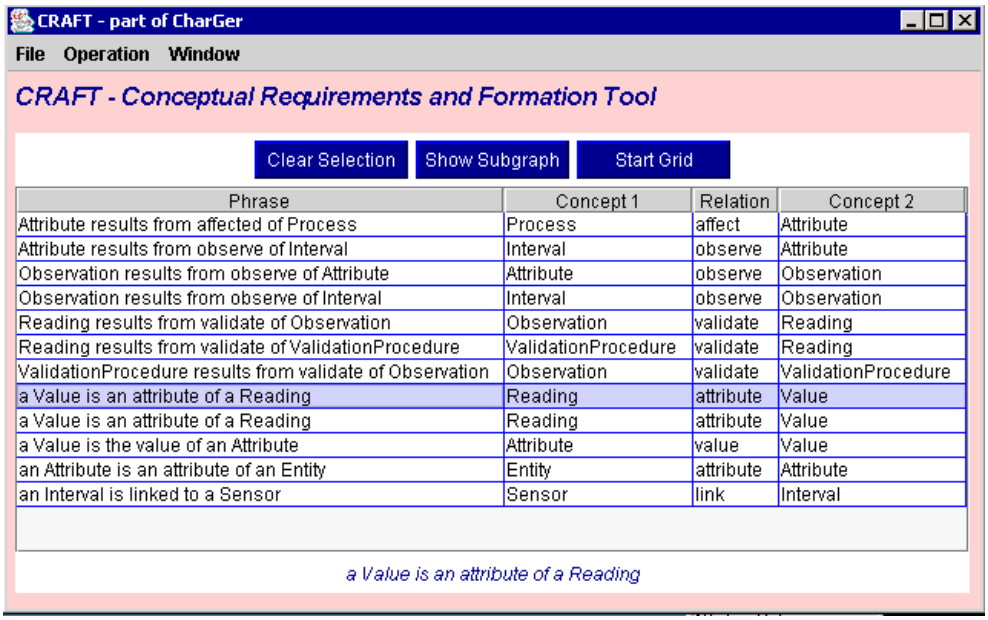


Figure 16. CRAFT main window.

The main window has the following buttons. Buttons are not shown unless they can be invoked.

CLEAR SELECTION

If a row is selected, then un-select it. Selecting more than one row is not recommended and probably will have no positive effect.

SHOW SUBGRAPH

Each phrase is based on a relationship in a conceptual graph in an open window. This button reveals the concepts and relations for the selected phrase by showing a selection box around them wherever they are present.

START GRID

Based on the selection, start a new repertory grid and begin acquiring rows and columns for the grid using the Grid Window.

The Grid Window

The repertory grid window is used to acquire and summarize knowledge through a repertory grid elicitation process. Consider this an experimental capability. The acquisition facility will acquire instances and attributes of whatever relation was chosen in the CRAFT window (see above). Choosing a relation in the CRAFT window selects a concept-relation-concept triple that will form the basis for the grid. **`Show Subgraph`** will remind the user of what relationship he/she chose.

Note that not all buttons will appear on the window at any given time; the buttons are set to appear only when they are applicable. Figure 17 shows an example of the repertory grid window.

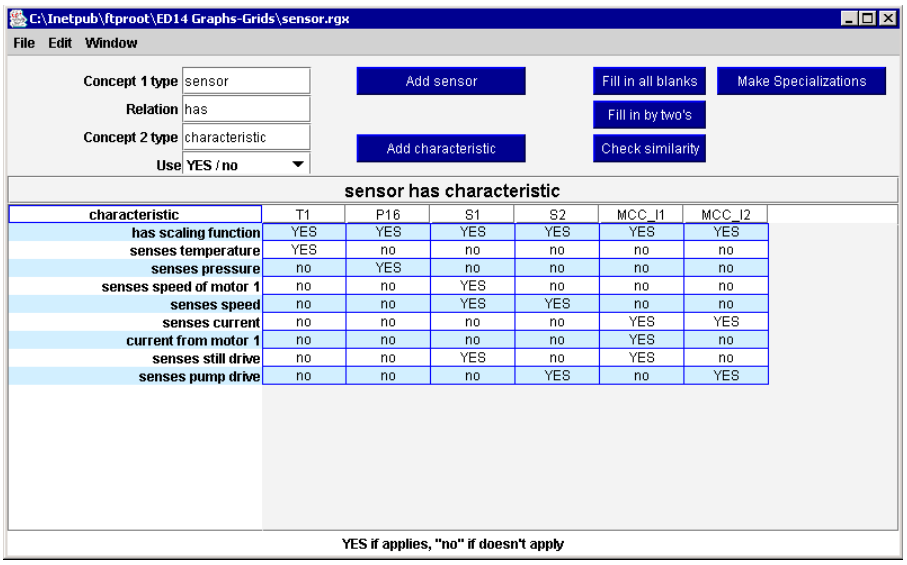


Figure 17. CRAFT repertory grid window.

CONCEPT 1 TYPE

This is the type label for the first concept in the relation. This label is filled in from the original concept-relation-concept triple.

RELATION

This is the relation label for the first concept in the relation. This label is filled in from the original concept-relation-concept triple.

CONCEPT 2 TYPE

This is the type label for the second concept in the relation. This label is filled in from the original concept-relation-concept triple.

ADD (INSTANCE OF CONCEPT 1)

Using the type label of concept 1, this button will allow you to add an instance of that concept to the grid as a column label. You may change it later by double-clicking on the column label.

SHOW SUBGRAPH

Takes the user back to the original conceptual graph and highlights the concept-relation-concept subgraph which forms the basis for this grid.

ADD (INSTANCE OF CONCEPT 2)

Using the type label of concept 2, this button will allow you to add an instance of that concept to the grid as a column label. You may change it later by double-clicking on the column label.

FILL IN ALL BLANKS

If there are any blank entries in the grid (cells), then this button will start a filling-in process that will consider each of the one by one, asking a grid question that should assist the user in providing an answer.

FILL IN BY TWO'S

Use a simplified dyadic elicitation technique to acquire new instances and attributes.

CHECK SIMILARITY

Look for columns with the same set of attributes. This will automatically invoke an acquisition process to acquire some new attribute(s) where the instances in those columns are different.

MAKE SPECIALIZATIONS

External Module Plug in Interface

Since Charger is freely available as open source software, you're free to distribute it under the terms of the LGPL license and of course add your own capabilities to be distributed freely under that license. Some users, however, may want to develop their own proprietary software using Charger. In that case, the architecture supports a facility known as a "module".

To prepare a module, first build a plugin "boot" class that implements the ModulePlugin interface (part of Charger's freely available code). This class then invokes whatever methods in whatever other packages you want in your external module.

The code in your module will have complete access to Charger and in general will need to be compiled with all the Charger packages in the class path. To make your module accessible to Charger, you create a module plugin class to provide the hooks to your module, one that controls the rest of your plugin operation, subject to the following constraints:

- It must be a class that is NOT inside any other package; i.e., it should appear at the top level of the jar file, or in the top level of some folder on the class path. (Your plugin's actual code package can be positioned anywhere you want.)
- The plugin class name must end with "ModulePlugin" (e.g., ABCModulePlugin.class).
- It must implement all the methods of the ModulePlugin interface. (see ModulePlugin.java)

Once you've done all that, your module name will appear on the tools menu. When selected, Charger will invoke the startup() method of your boot class. After that, it's up to you!

As a shortcut, you can invoke Charger from the command line with an argument. See above under Command Line Arguments.

Known Bugs and Restrictions

Most of these are meant to be handled in future versions. In the meantime, I hope you find CharGer to be useful in some way. Please let me know (delugach@cs.uah.edu) about other bugs.

KNOWN BUGS

- Undo does not work within a text field, although an entire editing operation can be undone once it is completed.
- Some CGIF files may cause a non-fatal, but perplexing Java parser error.

RESTRICTIONS

- Input and output routines generally depend on Java's Locale.ENGLISH settings, meaning that number formatting is in English conventions. Wherever possible, these dependencies have been removed, but some (e.g., graph dimensions) remain.
- Moving (or attempting to move) parts of a graph out of the current drawing area is not supported. Auto-scrolling is not supported.
- The actor plugin interface does not yet operate with contexts as input or output, although actors with such input and output contexts can be created, edited and saved. Inputs and outputs for operational actors must be simple concepts. The concepts themselves may be nested in contexts.
- The **`lookup`** actor uses databases that must all be located in the same folder; this is the folder named **`Databases`** within the top-level CharGer4 folder, or you may select a different one in the Actor Settings panel of the Preferences window. There is no pathname in the **`Database`** input concept to **`lookup`**. It is possible to work around this with aliases, links or shortcuts. It may be possible to play around with the actual referent of a **`Database`** concept to include a path, but I wouldn't rely on it, especially if graphs are to be used on more than one platform.
- The bottom symbol, ⊥ is represented by the type or referent "**`null`**" in referents or concepts. There is no provision for a ∀ or ∃ symbol in a referent. Charger follows the CGIF convention of using **`@every`** and **`@exists`**. The vertical bar "|" is otherwise illegal in a type or referent.
- In general, text in CharGer is case-sensitive, meaning that strings are compared "as is". The only exception is that when dealing with filenames, case sensitivity may depend on the conventions of the underlying platform operating system.
- Actors can only be activated by editing one of their input or output concept referents, or the actor name itself. There is no explicit activation of the actor on its own.
- Lambda expressions are not yet supported.
- Arithmetic with real numbers lacks precision; CharGer is not a reliable calculator for real numbers.
- When moving a context, all its contents go with it logically; if the move causes additional graph elements to appear enclosed visually, those additional elements are not logically part of the context!
- A concept can safely be a member of only one coreference set. It is not yet clear to this author (and others) how to interpret the semantics of a concept that might be a member of more than one coreference set.
- When joining or matching forms a new graph, elements of the new graph may overlap, since each set of elements is derived from a separate graph. The graph is stored internally in its correct form (as would be displayed by the CGIF format).
- Matching is not completely implemented. That is, several combinations and/or matching parameters will either not work at all or produce unpredictable results. The matching in CharGer is provided by Notio, which is currently undergoing additional development.
- Graph modality labels are for convenience only; no operational difference occurs in CharGer. (They can be turned on and off in the Compatibility panel of the Preferences window.)

- LINUX only: running CharGer will often show errors such as “cannot convert string …. To type VirtualBinding”. These are really Motif errors and can be safely ignored. If you are interested in the cause of the errors, see Google’s newsgroups and search on “cannot convert string” VirtualBinding and you’ll find out more.

Frequently Asked Questions

What does ■ mean on the linking lines?

Linking lines in conceptual graphs can be labeled. Generally relation arrows are to be numbered, although in many cases that's not necessary, since the types which are linked will serve to distinguish the arrows. The ■ is a selection handle merely for convenience in selecting a line for editing its label or deleting. If you click on the dot when deleting, then the arrow is deleted. If you click on the dot when you are editing text, you can change the relation label. The handle does not appear when printing.

To make the handles invisible (and make all lines unable to be selected!) uncheck the **Show line selection handle** option on the Preferences Panel.

How do I select a context?

Click somewhere on the *border* of the context. This is also how you select the context’s label for editing. This allows moving the context, deleting the context or editing its name, just as you would any other graph component.

How do I change the text label in a context?

Double-click somewhere on the *border* of the context. The context name should appear in an editing field and you can change it.

How do I delete an arc/arrow?

Choose the Delete Tool and then click on the arrow’s handle. If there is no handle showing, be sure to check the **Show line selection handle** option on the Preferences Panel.

How do I delete a concept/relation without deleting its linking arrows?

You can’t. CharGer cannot have a line unless there is an element at both of its ends. Deleting or moving a concept/relation also deletes or moves its lines.

How do I re-size a concept/actor/relation node?

There is no explicit way to just re-size a node. CharGer chooses the size of a node automatically, based on its text label, fonts and some additional cosmetic considerations. A context is expanded to enclose its contents. The “+” and “–“ keyboard hotkeys will enlarge or reduce one or more selected nodes one pixel at a time. Holding down the SHIFT key while pressing “+” or “–“ will enlarge or reduce the selected nodes more quickly. The “Shrink Selection” menu item will also change the size of nodes and contexts. A context can be expanded by moving its contents out toward its edges – the context border will expand to fully enclose them.

I opened a CGIF/CGX graph file and all the objects are crowded into the top left corner. Why?

Either a saved CGI graph did not have embedded CharGer layout information in its comments (see Compatibility Preferences panel) or a CGX file was created outside Charger and did not have any explicit layout information. Without this information, CharGer is unable to draw the graph on the screen, although its contents (semantics) should be preserved. To automatically lay out a graph on the screen, use the Auto-Layout feature in the Edit Menu.

How do I change a regular (non-negated) context to a "cut" and vice versa?

There is no way to do this directly. The only way to reverse the sense is to un-make the context/cut and then re-make it as a cut/context.

Is it possible to change the text color in all concepts or all relations at once?

The best way to do this is to create a new session default color set. This is done in the Appearnces Preferences panel. You can create a separate color scheme for each kind of node in a graph. Note that if the fill and text colors are the same, the text will seem to "disappear".

Technical Reference

CharGer XML File Format (version 3.1b and later)

As of version 3.1b, CharGer's stored graphs are in an XML format. This is a preliminary format, which unfortunately may change, but it serves as a starting point for others to develop compatible applications. Graphs in this form are suffixed with ".cgx" to distinguish them from the earlier versions.

The syntax of XML is beyond the scope of this manual. A DTD may be forthcoming, but in the meantime, this section describes the XML syntax.

The meaning of a "graph" in CharGer is a set of (possible unconnected) CharGer nodes, some of which may have links between them. The links are stored separately in the XML file.

The formal grammar for the file is as follows. Note that there must be space between characters unless they are special characters and that all parameter values within a tag must have double quotes around them. See general XML documentation for the syntax

CharGerXMLfile ::= xmlheader "<conceptualgraph" cgparmList ">" cgElementList "</conceptualgraph>"

xmlheader ::= "<?xml version="1.0" encoding="UTF-8"?>"

cgparmList ::= empty | cgparm | cgparm cgparmList

cgparm ::= "creator=" string | "version=" string | "created=" datestring | "modified=" datestring | "user=" string | "wrapLabels=" Boolean | "wrapColumns=" string

cgElementList ::= empty | cgElement | cgElement cgElementList

cgElement ::= cgNode | cgGraph | cgEdge

cgGraph ::= "<graph" objectParmList ">" objectTagList cgElementList "</graph>"

cgNode ::= "<" cgNodeName objectParmList ">" objectTagList "</"cgNodeName ">"

cgLine ::= "<" cgLineName objectParmList edgeParmList ">" objectTagList

"</"cgLineName ">"

cgNodeName ::= "concept" | "relation" | "actor" | "typelabel" |

"relationlabel" | "note"

cgEdgeName ::= "arrow" | "genspeclink" | "coref"

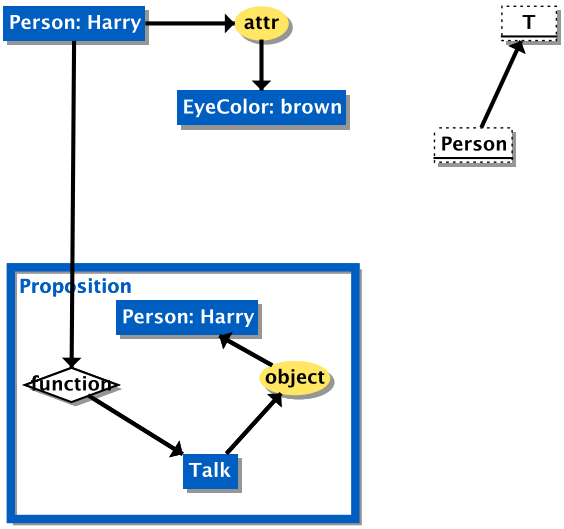
objectParmList ::= empty | objectParm | objectParm objectParmList
objectParm ::= “id=” idstring | “owner=” idstring | “label=” string | “negated=” boolean
boolean ::= “true” | “false”
edgeParmList ::= empty | edgeParm | edgeParm edgeParmList
edgeParm ::= “from=” idstring | “to=” idstring
idstring ::= string | “0”
objectTagList ::= empty | objectTag | objectTag objectTagList
objectTag ::= typeTag | referentTag | layoutTag | historyTag
typeTag ::= “<type>” typePartList “</type>”
typePartList ::= typePart | typePart typePartList
typePart ::= label | typeDescriptor
referentTag ::= “<referent>” referentPartList “</referent>”
referentPartList::= referentPart | referentPart referentPartList
referentPart::= label | referentDescriptor
referentDescriptor ::= string /* Note: currently undefined */
label ::= “<label>” string “</label>”
typeDescriptorPartList ::= typeDescriptorPart | typeDescriptorPart typeDescriptorPartList
typeDescriptor ::= wordnetDescriptor | genericDescriptor
wordnetDescriptor ::= “<wordnet-descriptor” wordnetParms “/>”
wordnetParms ::= “version=” version “pos=” partofspeech “offset=” uniqueoffset
partofspeech ::= “noun” | “verb” | “adjective” | “adverb”
uniqueoffset ::= positiveNumber
genericDescriptor ::= “<generic-descriptor” genericParms “>” typeDescriptorPartList “</generic-descriptor>”
genericParms ::= “pos=” partofspeech “definition=” string
layoutTag ::= “<layout>” layoutPartList “</layout>”
layoutPartList ::= empty | layoutPart | layoutPart layoutPartList
layoutPart ::= rectangle | color | font | edge
rectangle ::= “<rectangle” “x=” int “y=” int “width=” int “height=” int ““depth=” int “/>”
color ::= “<color” “foreground=” rgb “background=” rgb “/>”
rgb ::= int “,” int “,” int
font ::= “<font” “name=” fontname “style=” fontstyle “size=” fontsize “/>”
edge ::= “<edge” “arrowHeadWidth=” int “arrowHeadHeight=” int “edgeThickness” double
historyTag ::= “<history>” historyEventList “</history>”
historyEventList ::= empty | historyEvent | historyEventList
historyEvent ::= eventTag eventDescription
eventTag ::= “<event” eventClass timestamp eventType “>”
eventClass ::= “class=” + historyRecordClassname

timestamp ::= “timestamp=” timestampString
eventType ::= “type=” validtype (currently “FILE”, “USER”, “DERIVED” or “CUSTOM”)

There are certain constraints to make a well-formed graph for the CharGer parser, mostly due to the fact that it’s a one-pass parser.

- All owner objects must appear before any “owner=” parameters.
- An edge must appear after both of its endpoint objects have been listed (this can be a bit tricky with respect to contexts and coref lines; I’ve found it best to put corefs at the very end).
- If a parameter appears more than once, the last one will be accepted.
- The best rule of thumb is to provide for a specific order of the elements, generally graphs and nodes first, followed by arrows, finally followed by coreferent links (because they can cross graph boundaries)

Here is an example graph, whose file is shown below it:




```
<?xml version="1.0" encoding="UTF-8"?>
<conceptualgraph editor="CharGer" version="4.0.3" created="Sep 29, 2014 9:41:59 AM"
modified="Sep 29, 2014 9:45:31 AM" user="hsd" wrapLabels="true" wrapColumns="22">
<graph id="5494ec11:148c1dc59e9:-7ff3" owner="0">
    <type>
        <label>Proposition</label>
    </type>
    <layout>
        <rectangle x="5" y="5" width="1,200" height="900"/>
        <color foreground="0,94,192" background="0,94,192"/>
        <font name="SansSerif" style="1" size="14" />
    </layout>
    <graph id="5494ec11:148c1dc59e9:-7fe4" owner="5494ec11:148c1dc59e9:-7ff3">
        <type>
            <label>Proposition</label>
        </type>
```

```
        <layout>
            <rectangle x="78.5" y="289.25" width="259" height="189.5"/>
            <color foreground="0,94,192" background="0,94,192"/>
            <font name="SansSerif" style="1" size="14" />
        </layout>
        <relation id="5494ec11:148c1dc59e9:-7fed" owner="5494ec11:148c1dc59e9:-7fe4">
            <type>
                <label>object</label>
            </type>
            <layout>
                <rectangle x="264" y="360.5" width="52" height="25"/>
                <color foreground="0,0,0" background="255,231,100"/>
                <font name="SansSerif" style="1" size="14" />
            </layout>
        </relation>
        <actor id="5494ec11:148c1dc59e9:-7fec" owner="5494ec11:148c1dc59e9:-7fe4">
            <type>
                <label>function</label>
            </type>
            <layout>
                <rectangle x="92" y="365.5" width="68" height="25"/>
                <color foreground="0,0,0" background="255,255,255"/>
                <font name="SansSerif" style="1" size="14" />
            </layout>
        </actor>
        <concept id="5494ec11:148c1dc59e9:-7fef" owner="5494ec11:148c1dc59e9:-7fe4">
            <type>
                <label>Talk</label>
            </type>
            <layout>
                <rectangle x="208" y="428.5" width="40" height="25"/>
                <color foreground="255,255,255" background="0,94,192"/>
                <font name="SansSerif" style="1" size="14" />
            </layout>
        </concept>
        <concept id="5494ec11:148c1dc59e9:-7ff0" owner="5494ec11:148c1dc59e9:-7fe4">
            <type>
                <label>Person</label>
            </type>
            <referent>
                <label>Harry</label>
            </referent>
            <layout>
                <rectangle x="159" y="316.5" width="104" height="25"/>
                <color foreground="255,255,255" background="0,94,192"/>
                <font name="SansSerif" style="1" size="14" />
            </layout>
        </concept>
        <arrow    id="5494ec11:148c1dc59e9:-7fe6"    owner="5494ec11:148c1dc59e9:-7fe4"
label="-" from="5494ec11:148c1dc59e9:-7fef" to="5494ec11:148c1dc59e9:-7fed">
            <layout>
                <rectangle x="257.83" y="404.09" width="5" height="5"/>
                <color foreground="0,0,0" background="255,255,255"/>
                <font name="SansSerif" style="1" size="14" />
                <edge arrowHeadWidth="6" arrowHeadHeight="6" edgeThickness="1.5" />
            </layout>
        </arrow>
        <arrow    id="5494ec11:148c1dc59e9:-7fe5"    owner="5494ec11:148c1dc59e9:-7fe4"
label="-" from="5494ec11:148c1dc59e9:-7fed" to="5494ec11:148c1dc59e9:-7ff0">
            <layout>
                <rectangle x="251.98" y="350.94" width="5" height="5"/>
                <color foreground="0,0,0" background="255,255,255"/>
                <font name="SansSerif" style="1" size="14" />
```

```
                <edge arrowHeadWidth="6" arrowHeadHeight="6" edgeThickness="1.5" />
            </layout>
        </arrow>
        <arrow    id="5494ec11:148c1dc59e9:-7fe9"    owner="5494ec11:148c1dc59e9:-7fe4"
label="-" from="5494ec11:148c1dc59e9:-7fec" to="5494ec11:148c1dc59e9:-7fef">
            <layout>
                <rectangle x="170.88" y="404.95" width="5" height="5"/>
                <color foreground="0,0,0" background="255,255,255"/>
                <font name="SansSerif" style="1" size="14" />
                <edge arrowHeadWidth="6" arrowHeadHeight="6" edgeThickness="1.5" />
            </layout>
        </arrow>
    </graph>
    <relation id="5494ec11:148c1dc59e9:-7fee" owner="5494ec11:148c1dc59e9:-7ff3">
        <type>
            <label>attr</label>
        </type>
        <layout>
            <rectangle x="245.5" y="101.83" width="40" height="25"/>
            <color foreground="0,0,0" background="255,231,100"/>
            <font name="SansSerif" style="1" size="14" />
        </layout>
    </relation>
    <typelabel id="5494ec11:148c1dc59e9:-7feb" owner="5494ec11:148c1dc59e9:-7ff3">
        <type>
            <label>T</label>
        </type>
        <layout>
            <rectangle x="442" y="101.83" width="40" height="25"/>
            <color foreground="0,0,0" background="255,255,255"/>
            <font name="SansSerif" style="1" size="14" />
        </layout>
    </typelabel>
    <typelabel id="5494ec11:148c1dc59e9:-7fea" owner="5494ec11:148c1dc59e9:-7ff3">
        <type>
            <label>Person</label>
        </type>
        <layout>
            <rectangle x="392.5" y="190.5" width="57" height="25"/>
            <color foreground="0,0,0" background="255,255,255"/>
            <font name="SansSerif" style="1" size="14" />
        </layout>
    </typelabel>
    <concept id="5494ec11:148c1dc59e9:-7ff1" owner="5494ec11:148c1dc59e9:-7ff3">
        <type>
            <label>EyeColor</label>
        </type>
        <referent>
            <label>brown</label>
        </referent>
        <layout>
            <rectangle x="203.5" y="163.5" width="124" height="25"/>
            <color foreground="255,255,255" background="0,94,192"/>
            <font name="SansSerif" style="1" size="14" />
        </layout>
    </concept>
    <concept id="5494ec11:148c1dc59e9:-7ff2" owner="5494ec11:148c1dc59e9:-7ff3">
        <type>
            <label>Person</label>
        </type>
        <referent>
            <label>Harry</label>
        </referent>
```

```
        <layout>
            <rectangle x="76" y="101.83" width="104" height="25"/>
            <color foreground="255,255,255" background="0,94,192"/>
            <font name="SansSerif" style="1" size="14" />
        </layout>
    </concept>
    <genspeclink   id="5494ec11:148c1dc59e9:-7fe2"   owner="5494ec11:148c1dc59e9:-7ff3"
label="-" from="5494ec11:148c1dc59e9:-7fea" to="5494ec11:148c1dc59e9:-7feb">
        <layout>
            <rectangle x="439.5" y="156.67" width="5" height="5"/>
            <color foreground="0,0,0" background="255,255,255"/>
            <font name="SansSerif" style="1" size="14" />
            <edge arrowHeadWidth="6" arrowHeadHeight="6" edgeThickness="1.5" />
        </layout>
    </genspeclink>
    <arrow id="5494ec11:148c1dc59e9:-7fe3" owner="5494ec11:148c1dc59e9:-7ff3" label="-
" from="5494ec11:148c1dc59e9:-7ff2" to="5494ec11:148c1dc59e9:-7fec">
        <layout>
            <rectangle x="125" y="244.22" width="5" height="5"/>
            <color foreground="0,0,0" background="255,255,255"/>
            <font name="SansSerif" style="1" size="14" />
            <edge arrowHeadWidth="6" arrowHeadHeight="6" edgeThickness="1.5" />
        </layout>
    </arrow>
    <arrow id="5494ec11:148c1dc59e9:-7fe8" owner="5494ec11:148c1dc59e9:-7ff3" label="-
" from="5494ec11:148c1dc59e9:-7ff2" to="5494ec11:148c1dc59e9:-7fee">
        <layout>
            <rectangle x="211.04" y="112.33" width="5" height="5"/>
            <color foreground="0,0,0" background="255,255,255"/>
            <font name="SansSerif" style="1" size="14" />
            <edge arrowHeadWidth="6" arrowHeadHeight="6" edgeThickness="1.5" />
        </layout>
    </arrow>
    <arrow id="5494ec11:148c1dc59e9:-7fe7" owner="5494ec11:148c1dc59e9:-7ff3" label="-
" from="5494ec11:148c1dc59e9:-7fee" to="5494ec11:148c1dc59e9:-7ff1">
        <layout>
            <rectangle x="263.5" y="142.68" width="5" height="5"/>
            <color foreground="0,0,0" background="255,255,255"/>
            <font name="SansSerif" style="1" size="14" />
            <edge arrowHeadWidth="6" arrowHeadHeight="6" edgeThickness="1.5" />
        </layout>
    </arrow>
</graph>

</conceptualgraph>
```

Repertory Grid Proprietary File Format (v. 3.2b and later)

RepGridXMLfile ::= xmlheader “<repertorygrid>” rgheader rgattrList

rgelemList rgcellList “</repertorygrid >”

xmlheader ::= “<?xml version="1.0" encoding="UTF-8"?>”

rgheader ::= “<header>” headerLabelList “</header>”

headerLabelList ::= empty | headerLabel | headerLabel headerLabelList
headerLabel ::= “<headerlabel>”
cgparmList ::= empty | cgparm | cgparm cgparmList
cgparm ::= “creator=” string | “version=” string | “created=” datestring
cgElementList ::= empty | cgElement | cgElement cgElementList
cgElement ::= cgNode | cgGraph | cgEdge

CharGer Proprietary File Format (version 3.0b only)

See the .cg files to see what they look like. Edit them at your own risk! Here is a brief specification of the format. An example file is “catonmat.fact.cg” which looks like this:

```
CharGer|version=3.0b|creation=Feb 25, 2003 7:42:06 PM
Graph|17,0|Proposition|0,0,1000,1007|0,0,175|0,0,175
Concept|22,17|SIT|312,123,35,25|255,255,255|0,0,175
Relation|21,17|agent|203,224,55,25|0,0,0|255,231,100
Concept|20,17|Cat: Axel|63,123,113,25|255,255,255|0,0,175
Relation|19,17|location|385,243,80,25|0,0,0|255,231,100
Concept|23,17|MAT|488,123,45,25|255,255,255|0,0,175
Arrow|27,17|-|374,193,6,6|0,0,0|0,0,0|22,19
Arrow|26,17|-|171,183,6,6|0,0,0|0,0,0|21,20
Arrow|25,17|-|276,183,6,6|0,0,0|0,0,0|22,21
Arrow|24,17|-|464,193,6,6|0,0,0|0,0,0|19,23
\\
```

The formal grammar of the file is as follows:

The first line is of the form CharGer|version=<versionnumber>|creation=<date>

Succeeding lines are of the form

<ObjectType>|<ID>,<EnclosingID>|<TextLabel>|<Rectangle>|<textColor>|<fillColor>|<fromto ID>

- <ObjectType> Valid object types are: **`Graph`**, **`Concept`**, **`Relation`**, **`Actor`**, **`TypeLabel`**, **`RelationLabel`**, **`Arrow`**, **`Coref`**, and **`GenSpecLink`**. Spelling and capitalization must match exactly.
- <ID>,<EnclosingID> The first number <ID> is the unique identifier for this graph object. No two objects can have the same one in the same file. The second number <EnclosingID> is the unique identifier number for that object’s “owner”. In the 5th line of the example, Concept number 20, which is [Cat:Axel], is enclosed by Graph numbered 17. Note that Graph 17 itself has owner zero; that indicates this is the top-level graph in the file. If a context appears, it will have the type “Graph” and its owner will be whatever graph logically encloses it. There is no explicit provision for a graph to be a referent; its name constitutes its referent designator.
- <TextLabel> the text label for the object. For the top-level graph, it is often convenient to use the file’s name, although that is not required. Otherwise any text (except for a vertical bar and a newline) may appear in this term. Connecting lines may have a label, if so, it appears here.
- <Rectangle> the object’s display rectangle. Four numbers are required. They denote in order the upperleft corner’s x-coordinate, upper-left corner’s y-coordinate, width, and height. The

coordinate system used is Java's, where x increases going to the right and y increases going down.

- <textColor> a set of three integers representiing the RGB color value of this object's text. No alpha channel is supported for colors.
- <fillColor> a set of three integers representiing the RGB color value of this object's background fill. No alpha channel is supported for colors.
- For nodes in a graph (Graph, Concept, Relation, Actor and CGType) only five terms are present. For connecting lines in a graph (Arrow, Coref and GenSpecLink), a sixth term is present. It consists of two numbers, the first denoting the unique identifier of the source node, the second denoting the destination node. In the example, Arrow with ID 26 goes from ID 20 to ID 21, which means the arrow goes from the Relation "agent" to the Concept "Cat: Axel".

The file must be terminated by two backslashes "\\" on its own line.

CharGer Proprietary File Format (versions 2.6b or earlier)

CharGer can still read the old files (setting colors to the current default color scheme), but it can no longer save them in that form. All .cg files will be saved in the new format (see above). See the (old) .cg files to see what they look like. Edit them at your own risk! Here is a brief specification of the format. An example file is "catonmat.fact.cg" which looks like this:

```
Graph|12,0|catonmat.fact.cg|0,0,1000,1000|0,0,1000,1000
Concept|18,12|Cat: Albert|81,158,100,25|81,158,100,25
Concept|17,12|SIT|305,158,40,25|305,158,40,25
Concept|16,12|MAT|467,158,40,25|467,158,40,25
Relation|15,12|agent|191,263,56,18|191,263,56,18
Relation|14,12|location|357,263,77,18|357,263,77,18
Arrow|20,12|-|403,183,72,80|436,220,6,6|14,16
Arrow|19,12|-|141,183,70,80|173,220,6,6|15,18
Arrow|22,12|-|228,183,84,80|267,220,6,6|17,15
Arrow|21,12|-|333,183,55,80|357,220,6,6|17,14
\\
```

Each object in the graph is specified by a single line. Terms on each line are separated by a vertical bar.

- The first term is the graph object type. Valid types are: **`Graph`**, **`Concept`**, **`Relation`**, **`Actor`**, **`CGType`**, **`Arrow`**, **`Coref`**, and **`GenSpecLink`**. Spelling and capitalization must match exactly.
- The second term consists of a pair of numbers. The first number is the unique identifier for this graph object. No two objects can have the same one in the same file. The second number is the unique identifier number for that object's "owner". In the 2nd line of the example, Concept number 18, which is [Cat:Albert], is owned by Graph 12. Note that Graph 12 has owner zero; that indicates this is the top-level graph in the file. If a context appears, it will have the type "Graph" and its owner will be whatever graph logically encloses it. There is no explicit provision for a graph to be a referent; its name constitutes its referent designator.
- The third term is the text label for the object. For the top-level graph, it is often convenient to use the file's name, although that is not required. Otherwise any text (except for a vertical bar and a newline) may appear in this term. Connecting lines may have a label, if so, it appears here.

- The fourth term is the object's display rectangle. Four numbers are required. They denote in order the upperleft corner's x-coordinate, upper-left corner's y-coordinate, width, and height. The coordinate system used is Java's, where x increases going to the right and y increases going down.
- The fifth term is a relative rectangle, currently not used. Make it the same as the fourth term.
- For nodes in a graph (Graph, Concept, Relation, Actor and CGType) only five terms are present. For connecting lines in a graph (Arrow, Coref and GenSpecLink), a sixth term is present. It consists of two numbers, the first denoting the unique identifier of the source node, the second denoting the destination node. In the example, there is an arrow which goes from ID 15 to ID 18, which means the arrow goes from the Relation "agent" to the Concept "Cat: Albert".

The file must be terminated by two backslashes "\\" on their own line

Development Details

For any developers who are interested, the editor up to version 2.6b was developed under Metrowerks' CodeWarrior for the Macintosh. Since then, development has proceeded under Mac OS X using Project Builder, then Xcode, and now NetBeans. CharGer 4.0.4 consists of 221 Java classes in 24 packages, which make up approximately 57,000 lines of code (including comments). A description of these classes can be found at **`http://sourceforge.net/projects/charger/`**.

The Java console (i.e., the command line window) may display messages from time to time. In general, users can safely ignore them. If errors occur, the console may have information that can be useful in figuring out if there's a bug to report. Use the sourceforge ticket facility for reporting bugs to the developers.

Invoking *CharGer* from an application

If you have your own Java application (or possibly some other language's application) you may invoke CharGer quite easily, I think. The steps should be as follows:

- Make sure that the **`charger`** package is included in your Java project or environment.
- In your Java program, invoke the following call.

```
charger.Hub.setup();
```

- In the places where the driver needs to exit, the call

```
charger.Hub.closeOutAll();
```

That's all I had to do in my own main class. You may encounter problems; if so, report them and I'll do my best to figure them out.

In the future, interface calls will be provided to allow users to construct graphs in other programs and pass them into CharGer for editing.

Actor Plugin Interface

Starting with CharGer 2.6b, there's an actor plugin architecture whereby you can write your own actors that CharGer will incorporate. Of course, you're responsible for the reliability, etc. of the actors you write. If you want me to include them in the release (once you've tested them thoroughly!) send them to me for consideration.

The plugin interface provides a mechanism for creating external actors that can be incorporated into CharGer. Java classes that implement this interface are allowed to "plug in" to CharGer and show up in the actor list just as the primitive actors are. Classes implementing the ActorPlugin interface must appear in a package named "**plugin**". Responsibility for extracting referents from the **charger.Concept** arguments lies with the class implementing the interface; some convenience methods are found in **GraphUpdater**.

```
package charger;

public interface ActorPlugin
{
/**
      @return name by which the actor will be known throughout the system; this
is the string
            that will be used as the label on an actor in CharGer.
      */
      abstract public  String getPluginActorName();

/**
      Assumption about input and output vectors is that inputs are numbered
1..n and outputs
            are numbered n+1 .. m.
      @return List of input concepts (or graphs), each with a constraining type
(or "T" )
*/
      abstract public  Vector getPluginActorInputConceptVector();

/**
      Assumption about input and output vectors is that inputs are numbered
1..n and outputs
            are numbered n+1 .. m.
      @return List of output concepts (or graphs), each with a constraining
type (or "T" )
*/
      abstract public Vector getPluginActorOutputConceptVector();

/**
      @return Vector of objects, usually in string form, indicating other actor
constraints.
            Currently only "executable" and "commutative" are supported.
Attributes must include
            "executable" if you want CharGer to activate the actor.
*/
      abstract public Vector getPluginActorAttributes();


/**
      Perform the actor's function. Called by GraphUpdater when input or output
concepts change.
            @param inputs Vector of charger.Concept
            @return outputs Vector of charger.Concept
*/
    abstract public void performActorFunction( Vector inputs, Vector outputs );
```

```
/**
      Perform any clean-up required by the actor when it is deleted or its graph
            is de-activated.
*/
      abstract public void stopActor();

/**
      Give a string identifying the author, version, and email address of this
plugin
*/
      abstract public String getSourceInfo();
}
```