$\mathcal{F}$LORA-*2* Visualizer: User's Manual

Daniel Winkler[1] Michael Kifer[2]

[1]University of Innsbruck
Christoph-Probst-Platz
Innrain 52
6020 Innsbruck, Austria

[2]Department of Computer Science
State University of New York at Stony Brook
Stony Brook, NY 11794-4400, U.S.A.

June 12, 2007

Contents

1 Introduction

$\mathcal{F}$LORA-*2* Visualizer is an IDE environment for developing $\mathcal{F}$LORA-*2* applications. It provides the following components:

- a *text editor* for editing $\mathcal{F}$LORA-*2* source code
- a *visualizer* for viewing class hierarchies specified in $\mathcal{F}$LORA-*2*
- a *console* for communication with the $\mathcal{F}$LORA-*2* reasoner
- a *view* for the currently loaded modules

The following sections describe these components in detail and explain how to work with the visualizer to author and run $\mathcal{F}$LORA-*2* source files.

1.1 Technology

$\mathcal{F}$LORA-*2* Visualizer is based on the *Eclipse RCP* framework[1] and is therefore able to run on different operating systems, such as Windows, Linux and Mac OS.

$\mathcal{F}$LORA-*2* Visualizer requires that Java, version 1.5 or higher, is installed. It also requires $\mathcal{F}$LORA-*2* 0.95 (or higher)[2] and XSB 3.0.1 (or higher).[3]

For drawing the class hierarchy, $\mathcal{F}$LORA-*2* Visualizer relies on the graph drawing package JPowerGraph.[4] A jar-file of this package is included in the distribution and does not need to be downloaded separately.

1.2 Installation

$\mathcal{F}$LORA-*2* Visualizer is distributed both as a standalone program and as a plugin for Eclipse. The standalone version includes the necessary Eclipse libraries and is convenient if the user doesn't have an existing installation of Eclipse. The plugin version is much smaller and more flexible. It allows the user to reuse an existing installation of Eclipse and it can be upgraded more often. It also lets the user create standalone $\mathcal{F}$LORA-*2* Visualizer applications for use on other machines.

1. *Standalone version*:
   Download the zip file appropriate for your operating system from `http://flora.`

[1] Eclipse RCP is a platform for building and deploying rich client applications. It includes Equinox, a component framework based on the OSGi standard, the ability to deploy native GUI applications to a variety of desktop operating systems, such as Windows, Linux and Mac OS X, and an integrated update mechanism for deploying desktop applications from a central server.[Inc07]

[2] `http://flora.sourceforge.net`

[3] `http://xsb.sourceforge.net`

[4] `http://sourceforge.net/projects/jpowergraph/`

`sourceforge.net/download.php` and extract the files into some directory. The executable will then be found in the *eclipse* subdirectory. On Windows, it is called FLORA2VISUALIZER.EXE and on on Unix-based systems FLORA2VISUALIZER.

2. *Update Site*:
   If you have an existing installation of Eclipse, you can install $\mathcal{F}$LORA-*2* Visualizer from the update site. This version is typically much smaller than the standalone version, as it does not include Eclipse libraries. If no standalone version of the $\mathcal{F}$LORA-*2* Visualizer exists for your OS, this (and CVS) would also be your only option.

   To install $\mathcal{F}$LORA-*2* Visualizer from the update site, select *Software Updates* → *Find and Install* from the Help menu. In the resulting dialog, choose *Search for new features* and click Next. In the next dialog, click *New Remote Site* and add the following URL:

   `http://flora.sourceforge.net/visualizer/`

   To use the visualizer, you should start Eclipse and choose $\mathcal{F}$LORA-*2* Visualizer Perspective from the *Windows* menu by selecting the item *Open Perspective* → *Other*. Then choose *Flora* and press *OK*.

3. *CVS version*:
   The CVS version is for people who want to use the latest development versions of the visualizer without waiting for official releases. Such versions might include non-critical bug fixes and new features that are not available in the official releases. This is also the version to use for the developers of the $\mathcal{F}$LORA-*2* Visualizer.

   To install this version, you need to download the source code of $\mathcal{F}$LORA-*2* Visualizer from CVS at *flora.cvs.sourceforge.net*. For information on how to access the $\mathcal{F}$LORA-*2* CVS see `http://sourceforge.net/cvs/?group_id=50604` (check out the CVS module *flora2-visualizer*).

   The checked out files of the visualizer must be imported as an eclipse project. The easiest way to do this is to use the Eclipse *CVS Repository Exploring* perspective, since the files automatically get imported as projects when checked out from CVS. Alternatively you can get the files checked out of the CVS repository manually and then import them to Eclipse. Note that each *net.sourceforge.flora** directory must be a separate project. This is done with the Eclipse import wizard by selecting the following item from the *File* menu: *Import* → *Existing Projects into Workspace*.

   Next, you should create an eclipse plugin or a standalone application. The latter might be useful if you want to, for example, use it on a computer that does not have an Eclipse installation. This is done as follows:

   - A developer's plugin
     Open the product file of the *net.sourceforge.flora* project (called *flora.product*). At the *Overview* tab you will find two links:
     - Launch the product

– Launch the product in Debug mode

If you click one of these, a new launch configuration will be created and the $\mathcal{F}$LORA-*2* Visualizer will start.

If the launch failed, you need to update the product configuration. Choose the *Configuration* tab of the product configuration and click *Remove All*. Then click the *Add* button and add all *net.sourceforge.flora** projects. Then click the *Add Required Plug-ins* button, save the configuration and try again. Once the configuration is set up properly, the application can be launched using the *Run* menu item or the run/debug buttons on the toolbar.

- A standalone application
  Before creating a standalone application, check if everything is configured properly by starting a previously configured developer's plugin, as described above. If everything is fine, you can export the project as a standalone application. To do so, use the *Eclipse Product export page* link at the *Overview* tab. This opens a wizard which lets you choose whether you want to create the standalone application as a *Directory* or as an *Archive File*.
  Additionally there is an option to export the application for different OS platforms. To use this option, you must have the *RCP delta pack* installed on your system; it can be downloaded from `http://download.eclipse.org/eclipse/downloads/`.
- A update site
  One would create this type of a plugin, if you already have Eclipse on your system. A developer would also use this option to create a version of the visualizer for the update site.

  As with a standalone application, check if everything is configured properly by starting the previously configured developer's plugin. If everything is fine, you can export the project as a plugin for the update site. To do so open the *net.sourceforge.flora.eclipse.updatesite* project. Then click the *Build All* button on the *Site Map* tab. Place the resulting file on the Web site of your choice and add the site to Eclipse by selecting the following item from the *Help* menu: *Software Updates* → *Find and Install...* → *Search for new features to install* → *Add Local Site...*[5]

2 Configuring the $\mathcal{F}$LORA-*2* Visualizer **Reasoner**

The $\mathcal{F}$LORA-*2* Visualizer connects to $\mathcal{F}$LORA-*2* through Interprolog[6] and the $\mathcal{F}$LORA-*2* *javaAPI*. In order to function properly, you need to configure the visualizer and point it to your installation of $\mathcal{F}$LORA-*2*. This is the main configuration step. Other steps are optional. They will default to the standard choices, if the user does not change them.

[5] `http://wiki.eclipse.org/index.php/FAQ_How_do_I_create_an_update_site_%28site.xml%29%3F`
[6] `http://www.declarativa.com/interprolog/`

To configure the visualizer to use your installed version of $\mathcal{F}$LORA-*2*, choose the following menu item: *Window* → *Preferences*. At the preference page you can specify the following:

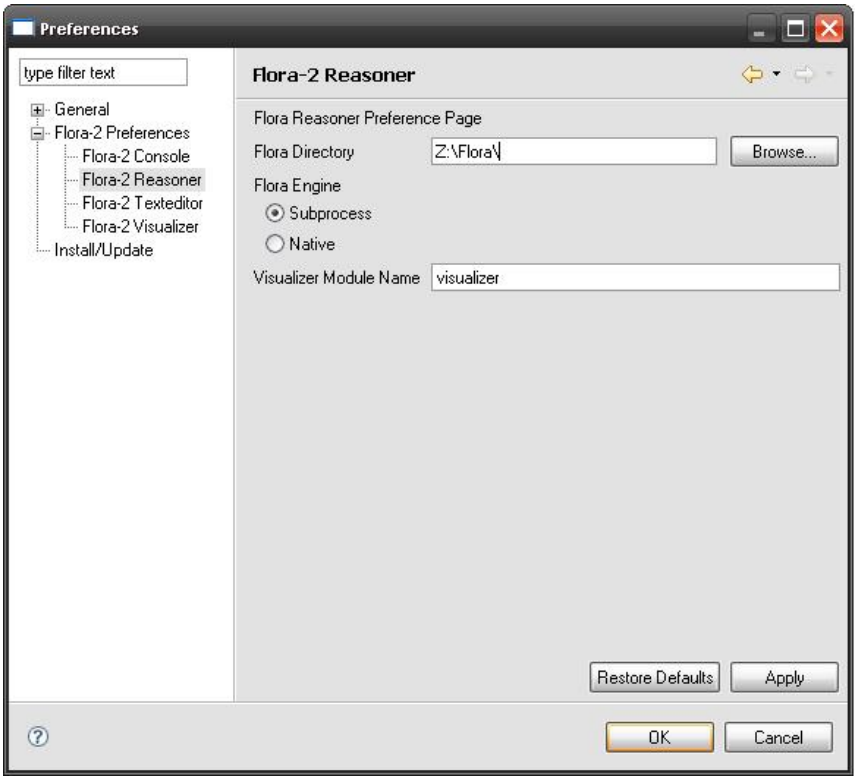

Figure 1: $\mathcal{F}$LORA-*2* Reasoner Preference Page

- *Flora Directory*
  The directory where $\mathcal{F}$LORA-*2* is located.

- *Interprolog Engine*
  Interprolog provides two engines: *Native* and *Subprocess*. The native engine is faster, but it does not work well under Linux.

- *Visualizer Module Name*
  The name of the $\mathcal{F}$LORA-*2* module through which the reasoner communicates with the visualizer. You can leave the default unchanged unless the same name is already used in your $\mathcal{F}$LORA-*2* application.

After applying the changes the visualizer is ready for use. The user can talk to the $\mathcal{F}$LORA-*2* reasoner through $\mathcal{F}$LORA-*2* Visualizer Console and also (in a limited way) by directly manipulating the objects in $\mathcal{F}$LORA-*2* Visualizer Class Diagram View. We describe these components below.

3 Working with the Visualizer

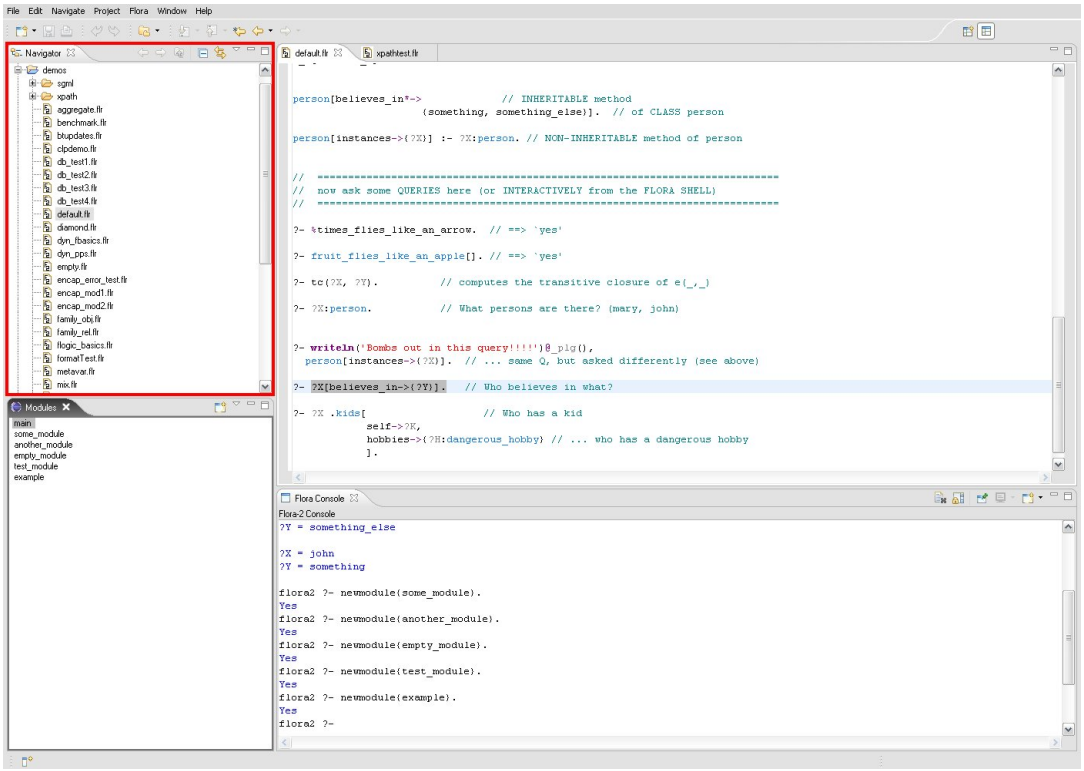

Figure 2: $\mathcal{F}$LORA-*2* Visualizer Navigator

To begin working with $\mathcal{F}$LORA-*2* Visualizer, Eclipse (or the $\mathcal{F}$LORA-*2* Visualizer standalone application) has to be started. You will be asked to select a directory for a workspace. This location specifies where the projects and preferences will be stored. To create a new project, select *New* → *Project* from the file menu. A dialog will appear and prompt for the name of a new project. For more information have a look at the Eclipse help[7].

For each project, Eclipse creates a subdirectory in the directory of the workspace; it has the same name as the project. All project's files will be stored in this directory.

Once the project is created, you can either import existing $\mathcal{F}$LORA-*2* files or use $\mathcal{F}$LORA-*2* Visualizer to create new knowledge bases. Files can be imported by choosing the *File* → *Import* menu item. A dialog will appear and lead you through the process of importing files to the project. The imported files will then be copied to your project directory.

Note that $\mathcal{F}$LORA-*2* Visualizer will work *only* with the copies of the files stored in the project subdirectory. If these files were created via import and you modify the source files

[7] `http://help.eclipse.org/help32/index.jsp?topic=/org.eclipse.platform.doc.user/gettingStarted/qs-07a.htm`

(from which the import was created) then these changes will *not* be propagated to the project.

Also, if you copy a file into a project directory *manually* outside of Eclipse then this new file will not automatically be made part of the project. To include it in the project, you need to use the mouse to select the requisite files in the $\mathcal{F}$LORA-*2* Visualizer Navigator View and then press *F5*. For more information on importing, have a look at the Eclipse help[8].

To create a new file, choose *New* → *Other...* → *File* from the *File* menu. Then give the name to the new file (do not forget the file ending e.g. *new_file.flr*) and choose a project.[9] The files can then be edited with the $\mathcal{F}$LORA-*2* Visualizer Text Editor. See Section 6 to learn more about the editor.

After creating the necessary $\mathcal{F}$LORA-*2* modules, we can start working with the visualizer. The system allows the user to load, run, and query $\mathcal{F}$LORA-*2* knowledge bases through $\mathcal{F}$LORA-*2* Visualizer Console and to visualize class diagrams and object definitions. These capabilities are described in the following sections.

To learn more about the Eclipse workbench see the *Workbench User Guide* at `http://help.eclipse.org`.

4 The $\mathcal{F}$LORA-*2* Visualizer Console

The $\mathcal{F}$LORA-*2* Visualizer Console allows you to communicate with the current $\mathcal{F}$LORA-*2* session. All $\mathcal{F}$LORA-*2* commands and queries are available through the console.

4.1 Configuration

It is not necessary to configure the console unless you dislike the colors with which the visualizer highlights text. To change the colors, choose the $\mathcal{F}$LORA-*2* Console Preference page where you can alter the defaults:

- *Default color*
  This is used to highlight user input.
- *Response color*
  This color is used for normal $\mathcal{F}$LORA-*2* feedback.
- *Error color*
  This color is used for error messages.

[8] `http://help.eclipse.org/help32/topic/org.eclipse.platform.doc.user/gettingStarted/qs-31a.htm`

[9] `http://help.eclipse.org/help32/topic/org.eclipse.platform.doc.user/reference/ref-39.htm`

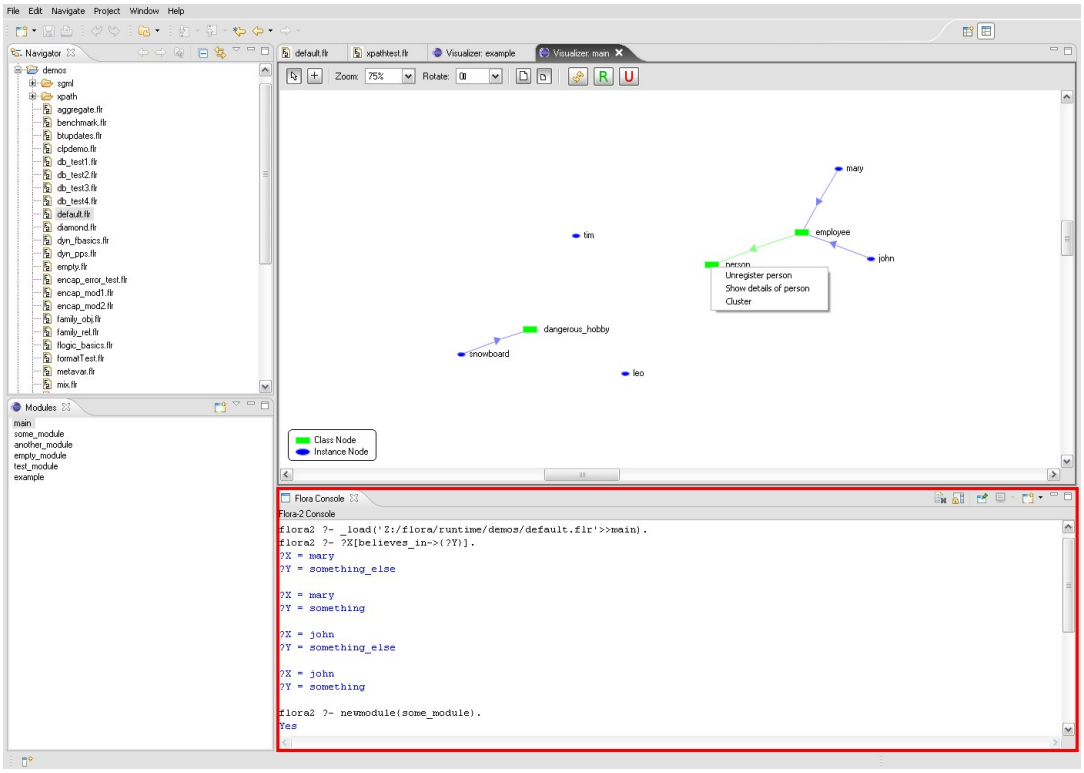

Figure 3: $\mathcal{F}$LORA-*2* Visualizer Console View

4.2 Usage

The $\mathcal{F}$LORA-*2* Visualizer Console parses user commands and sends them to the $\mathcal{F}$LORA-*2* Visualizer Reasoner. The console then displays the results, errors, and other possible feedback. You may also see some commands that you did not type: these are issued by the visualizer.

5 The $\mathcal{F}$LORA-*2* Visualizer **Module View**

The $\mathcal{F}$LORA-*2* Visualizer Module View shows all currently loaded $\mathcal{F}$LORA-*2* modules (except the module which is used for the visualizer itself). This view does not have configuration options.

5.1 Usage

The module view supports creation, deletion, and hiding of $\mathcal{F}$LORA-*2* modules. It also supports drag-and-drop idioms for loading and adding $\mathcal{F}$LORA-*2* knowledge to modules.

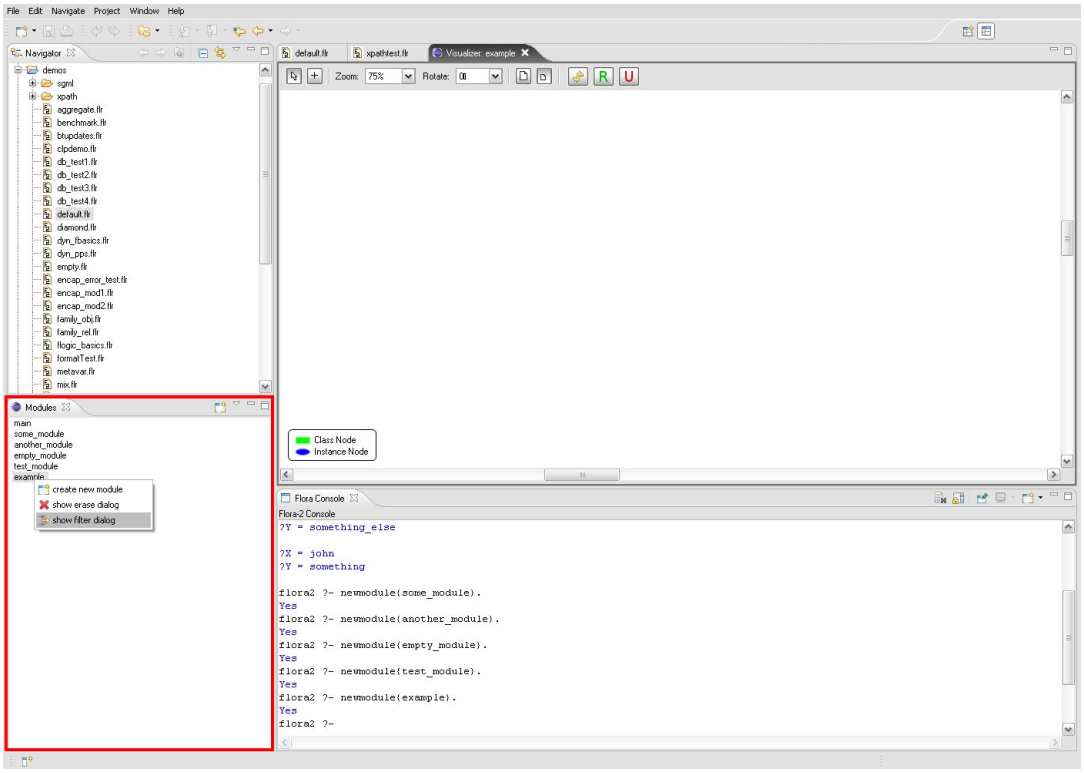

Figure 4: $\mathcal{F}$LORA-*2* Visualizer Module View

5.1.1 Loading and Adding Files

$\mathcal{F}$LORA-*2* files imported into a visualizer project appear in the Navigator View of the $\mathcal{F}$LORA-*2* Visualizer. Any such file can be dragged with the mouse into any of the modules shown in the Module View. When you drop such a file into a module, a context menu pops up and you can select one of the following operations:

- *load to module and erase registered objects*
  This option erases all registered object and loads the file to the specified module. The current contents of the module is erased as well. This corresponds to the `_load` command in $\mathcal{F}$LORA-*2*.

  The registered objects mentioned here are the objects and classes that are supposed to be displayed in the $\mathcal{F}$LORA-*2* Visualizer Class Diagram View. See Section 7.2.1 for more details on the concept of registered objects.

- *load to module and keep registered objects*
  This option loads the file into the specified module and erases the module's earlier contents, but it preserves the registered objects. This operation is useful when you change a $\mathcal{F}$LORA-*2* file and want to reload it into the same module where it was before. In that case, the class diagram for that module will be updated.

- *add to module and keep registered objects*
  This action corresponds to the `_add` command of *FLORA-2*. It adds knowledge to an existing module without erasing the old contents. The registered objects are preserved and the class diagram is updated.

5.1.2 Creating New *FLORA-2* Modules

To create a new *FLORA-2* module using the *FLORA-2* Visualizer Module View, click the *create new module* button or use the context menu, which pops up when you right-click on the list of modules. This causes a display of a dialog where you can specify the name of a module to be created. Click *OK* to confirm the request. This is equivalent to the `newmodule` command.

5.1.3 Erasing *FLORA-2* Modules

To erase an existing module using the Module View, choose *show erase dialog* from the context menu, which pops up when you right-click over the module list. In the dialog that is then displayed, you can check the modules to be erased and then click the OK emm button. This is equivalent to the `erasemodule` command.

5.1.4 Hiding *FLORA-2* modules

To hide selected *FLORA-2* modules from the *FLORA-2* Visualizer Module View, use the *show filter dialog* in the context menu. he dialog box that is then displayed allows the user to check the modules to be hidden. Note that this does not erase those modules—just hides them from the view.

6 The *FLORA-2* Visualizer Text Editor

The *FLORA-2* Visualizer Text Editor is used to edit *.flr* files. It supports syntax-highlighting, auto-indentation, and text-formatting.

6.1 Configuration

There is no need to configure the text editor unless you dislike its default settings. If you decide to change those settings, the *FLORA-2* Visualizer Text Editor configuration page is located at the *Flora-2 Preferences* → *Flora-2 Text Editor* menu item. The preference page allows the user to change the following settings:

- *Tab width*
  The number of space characters which are displayed in place of a tab character.

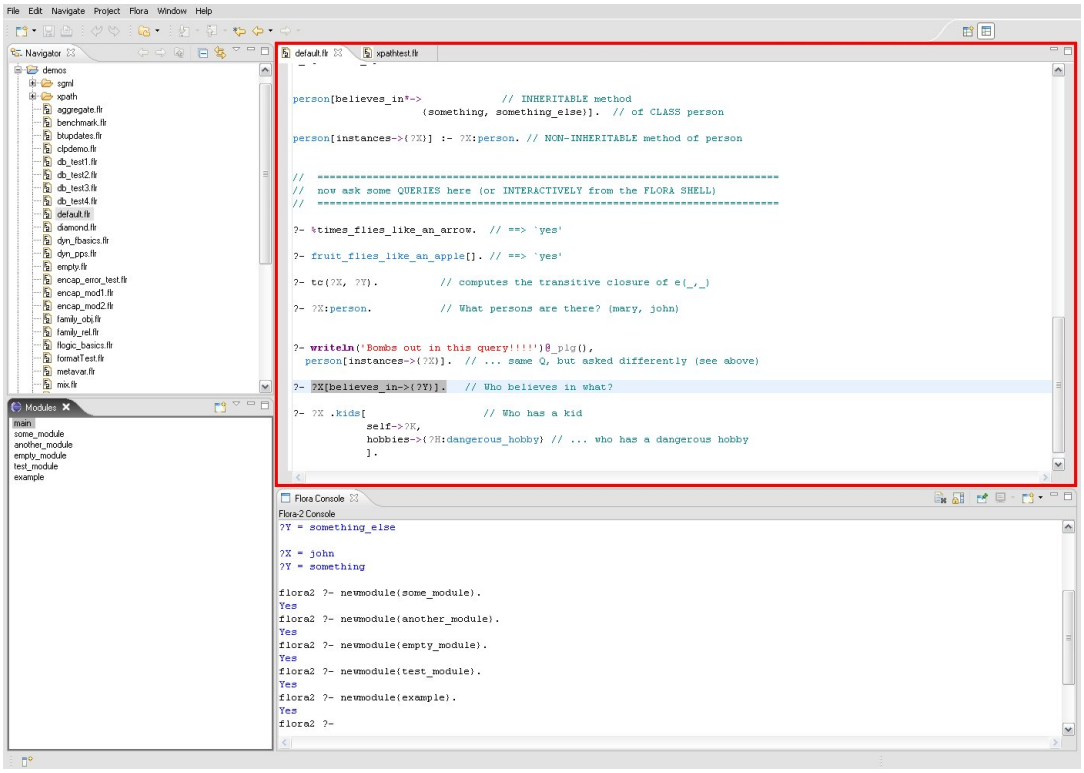

Figure 5: $\mathcal{F}$LORA-*2* Visualizer Text Editor

- *Tab type*
  If set to *Tab*, the $\mathcal{F}$LORA-*2* Visualizer Text Editor displays '\t' in place of a tab. If set to *Space*, the tab is displayed as a sequence of spaces. The number of the spaces is controlled by the aforesaid tab width parameter.

- *Colors*
  This options allow to change the colors of different text tokens: strings, numbers, keywords, etc.

6.2 Usage

The $\mathcal{F}$LORA-*2* Visualizer Text Editor is specifically designed for the $\mathcal{F}$LORA-*2* syntax. It automatically highlights the various syntactic components and it indents text items to their proper position when the user hits the *Return* key. Also, when the user hits the *TAB* key then the line marked by the cursor is also indented in accordance with the $\mathcal{F}$LORA-*2* syntactic conventions.

6.3 Menu

When $\mathcal{F}$LORA-*2* Visualizer text editor is active, $\mathcal{F}$LORA-*2* menu is extended with the following actions:

- *Format*
  This action formats the highlighted (or the whole if nothing is highlighted) text. This corrects the indentations and beautifies the spaces between tokens.
- *Open Buffer*
  This action opens a new blank buffer in the editor and allows the user to start typing new text. This text can be saved as usual, through the *File* → *Save as* menu item.
- *Add Region to Module*
  This action adds the rules and the facts in the selected text region of the currently active $\mathcal{F}$LORA-*2* Visualizer Text Editor to a $\mathcal{F}$LORA-*2* module. To specify the module, a dialog pops up.
- *Add Region*
  This action adds the rules/facts of the selected text region of the current $\mathcal{F}$LORA-*2* Visualizer Text Editor to $\mathcal{F}$LORA-*2*'s module `main`.
- *Load Region to Module*
  This action loads the rules and facts in the selected text region of the currently active $\mathcal{F}$LORA-*2* Visualizer Text Editor to a $\mathcal{F}$LORA-*2* module. A dialog pops up where the user can specify the desired module.
- *Load Region*
  This action adds the contents of the selected text region of the currently active Text Editor to the $\mathcal{F}$LORA-*2* module `main`.
- *Execute Region as a Query*
  This action executes the selected text of the current $\mathcal{F}$LORA-*2* Visualizer Text Editor as a query.

7 The $\mathcal{F}$LORA-*2* Visualizer Class Diagram View

The $\mathcal{F}$LORA-*2* Visualizer Class Diagram View shows the class hierarchy of the selected $\mathcal{F}$LORA-*2* module. Only the registered objects are displayed. The concept of registered objects is explained in Section 7.2.1.

7.1 Configuration

The class diagram view does not need to be configured. But the following parameters can still be changed though the page accessible by selecting *Flora-2 Preferences* → *Flora-2 Visualizer*.

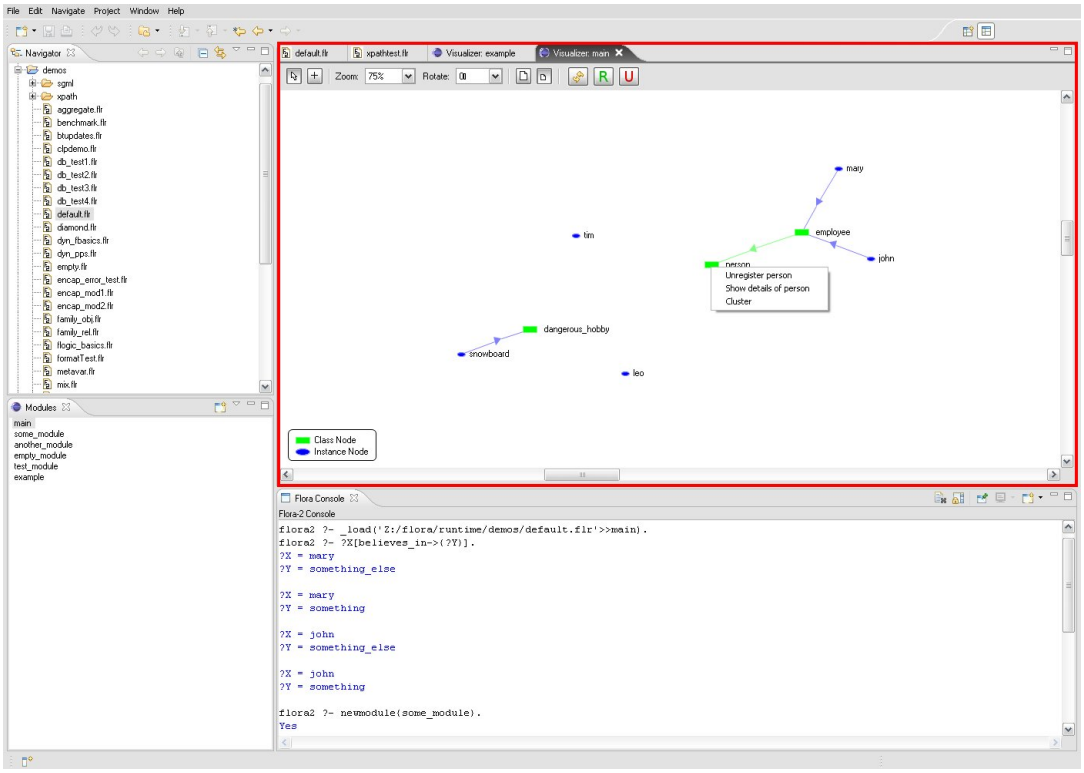

Figure 6: $\mathcal{F}$LORA-*2* Visualizer Class Diagram View

- *Class/instance Node Colors*
- *Zoom level*
  This allows the user to zoom the class diagram in and out.
- *Node Size*
  The size of the nodes that represent classes and instances.
- *Use Instance Cluster*
  If checked the graph makes use of instance clusters. This means that the graph will contain one big blob to represent all instances of each class, if the number of instances exceeds the cluster size (see next). This is useful when the graph can have too many instances.
- *Minimum Cluster Size*
  The number of instances in a class at which the graph shows an instance cluster instead of individual instances for the class. The $\mathcal{F}$LORA-*2* Visualizer Class Diagram View allows one to dynamically switch between clustered and unclustered views.

7.2 Usage

Each $\mathcal{F}$LORA-*2* Visualizer Class Diagram View shows a class hierarchy (restricted to registered objects) for a particular $\mathcal{F}$LORA-*2* module. A $\mathcal{F}$LORA-*2* Visualizer Class Diagram View for a module can be opened by clicking on the module in the $\mathcal{F}$LORA-*2* Visualizer Module View (see Section 5).

7.2.1 Registering Objects for Display in Class Diagram Views

In general, a $\mathcal{F}$LORA-*2* knowledge base can have an infinite number of objects and classes. Clearly, no graphical display can show this kind of hierarchies. However, in most cases users want to focus only on a small subset of the hierarchy. $\mathcal{F}$LORA-*2* Visualizer lets users specify this focus through the mechanism of *registration*.

There are two ways to register objects for display. The simplest one is to use the Class Diagram View, which provides the *R* and *U* buttons for registering and deregistering objects. This interface is quite effective for quick experiments, but it is too weak and inflexible, since it requires that the user provide the list of objects to be registered.

A more flexible and permanent mechanism is to use the `%register` and `%unregister` methods of the visualizer module. For instance, to register objects *a*, *b*, and *c*, in a $\mathcal{F}$LORA-*2* module `foobar` one could put the following query in a $\mathcal{F}$LORA-*2* program (or to execute it at command prompt):

```
?- foobar[%register([a,b,c])]@visualizer.
```

Here we assume that $\mathcal{F}$LORA-*2* Visualizer uses the module called `visualizer` to communicate with $\mathcal{F}$LORA-*2*. This module is specified when you configure $\mathcal{F}$LORA-*2* Visualizer, as described in Section 2. If this is specified then the user will not need to register the corresponding objects by clicking the *R* button each time the visualizer starts.

There is an even more powerful way of specifying which objects to register. To this end, the user can define a *unary* predicate, say `myregobjs`, and define it via the usual $\mathcal{F}$LORA-*2* deductive rules. It is user's responsibility, however, to ensure that the number of objects thus specified is finite (otherwise, the visualizer will hang). Then the user can include the following query in the source file:

```
?- foobar[%register(${myregobjs(?)})]@visualizer.
```

This query will register every object that is returned as an answer to the query

```
?- myregobjs(?X).
```

The method `%unregister` can be used to remove objects from the Class Diagram View. It also can take a list of objects or a unary predicate.

7.2.2 Context Menu

The nodes in the Class Diagram View display context menus when one right-clicks on them. The context menu contains the following items:

- *Unregister*
  This action deregisters the object under the context menu.
- *Show details*
  This action opens a *Details Dialog* with two panes showing the values and types of the attributes of the object under the menu. For nodes that represent instance-clusters, right-clicking opens a dialog listing all instances in the cluster. You can double-click on an instance to open the *Details Dialog* for that object. The *Details Dialog* also pops up when one double-clicks on an node that represents an object.
- *Cluster*
  This action applies only to $\mathcal{F}$LORA-*2* class nodes. It redraws the graph and clusters the instances of that class.
- *Uncluster*
  This action applies only to $\mathcal{F}$LORA-*2* class nodes and to cluster nodes. It redraws the graph and shows individual instances of the class instead of a single cluster node.

It should be noted that some commands executed from the console may affect the class diagram but $\mathcal{F}$LORA-*2* Visualizer might not be able to detect this. For this purpose, the Class Diagram View provides a *Redraw* button.

ACKNOWLEDGEMENTS. We would like to thank Mick Kerrigan for developing JPowerGraph and for helping along with this project.

References

[Inc07] Eclipse Foundation Inc. Rich client platform. Technical report, Eclipse Foundation Inc., 2007. Available online at `http://www.eclipse.org/home/categories/rcp.php`; visited on April 25th 2007.