Overview of the Garnet System

Brad A. Myers
Andrew Mickish

October 1993

Abstract

This article provides an overview of the Garnet system, and a guide to this set of manuals.

This research was sponsored by the Avionics Lab, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U. S. Air Force, Wright-Patterson AFB, OH 45433-6543 under Contract F33615-90-C-1465, Arpa Order No. 7597. Additional support for Garnet has been provided by NEC, Apple, Adobe, and General Electric.

1. Introduction

The Garnet research project in the School of Computer Science at Carnegie Mellon University is creating a comprehensive set of tools which make it significantly easier to create graphical, highly-interactive user interfaces. The lower levels of Garnet are called the ‘‘Garnet Toolkit,’’ and these provide mechanisms that allow programmers to code user interfaces much more easily. The higher level tools allow both programmers and non-programmers to create user interfaces by just drawing pictures of what the interface should look like. Garnet stands for Generating an Amalgam of Real-time, Novel Editors and Toolkits.

At the time of this writing, the Garnet toolkit is in use by about 100 people in 50 projects throughout the world. This document contains an overview, tutorial, and a full set of reference manuals for the Garnet System.

This manual describes version 2.2 of Garnet. It replaces all previous versions and the change documents for versions 1.3, 1.4, 2.0, 2.1, and 2.2.

Garnet is written in Common Lisp for the X window manager. Therefore, Garnet is quite portable to various environments. It works in virtually any Common Lisp environmnt, including Allegro (Franz), Lucid, CMU, Harlequin, CLISP and AKCL Common Lisps. The workstations we know about it running on include Sun, DEC, HP, Apollo, IBM RT, IBM 6000, TI, SGI, and NeXTs running X/11, but there may be others. Currently, Garnet supports X/11 R4 or R5 using the standard CLX interface. Garnet does *not* use the standard Common Lisp Object System (CLOS) or any X toolkit (such as Xtk or CLIM).

In the near future, we hope to have a version of Garnet running on the Macintosh under the Macintosh operating system using Apple CommonLisp. When this is ready, applications using Garnet will not need to be changed to run on either X or the Macintosh.

This document is a technical reference manual for the entire Garnet system. There have been many conference and journal papers about Garnet (see section 15 for a complete bibliography). The best overview of Garnet is [Myers 90]. Section 15 includes instructions for retrieving some Garnet papers via FTP.

2. Garnet Bulletin Board

There is an international bboard for Garnet users named `comp.windows.garnet`. Topics discussed on this bboard include user questions and software releases. There is also a mailing list called `garnet-users@cs.cmu.edu` which carries exactly the same messages as the bboard. If you cannot read the bboard in your area, please send mail to `garnet-users-request@cs.cmu.edu` to get on the mailing list.

You can report bugs to `garnet-bugs@cs.cmu.edu` which is read only by the Garnet developers.

3. Important Features of Garnet

Garnet is being designed as part of a research project, so it contains a number of novel and unique features. Some of these are:

- The Lapidary tool is the only interactive tool that allows application-specific graphics and new widgets to be created without programming.
- The Garnet Toolkit is designed to support the *entire* user interface of an application; both the contents of the application window and its menus and dialog boxes. For example, Garnet directly supports selecting graphical objects with the mouse, moving them around, and changing their size.
- It is *look-and-feel independent*. Garnet allows the programmer to define a new graphical

style, and use that throughout a system. Alternatively, a pre-defined or standard style can be used, if desired.

- It uses a *prototype-instance* object model instead of the more conventional class-instance model, so that the programmer can create a *prototype* of a part of the interface, and then create instances of it. If the prototype is changed, then the instances are updated automatically. Therefore, Garnet uses its own, custom object system (called KR), and does not use CLOS.

- *Constraints* are integrated with the object system, so any slot (also called an ‘‘instance variable’’) of any object can contain a formula rather than a value. When a value that the formula references changes, the formula is re-evaluated automatically. Constraints can be used to keep lines attached to boxes, labels centered at the top of rectangles, etc. (see Figure 1). Constraints can also be used to keep application-specific values connected with the values of graphical objects, menus, scroll bars or gauges in the user interface.

- Objects are *automatically refreshed* when they change. Pictures are displayed by creating graphical objects which are retained. If a slot of an object is changed, the system automatically redraws the object and any other objects that overlap it. Also, the system handles window refresh requests from X.

- The programmer specifies the handling of input from the user at a high level using abstract *interactor* objects. Typical user interface behaviors are encapsulated into a few different types of interactors, and the programmer need only supply a few parameters to get objects to respond to the mouse and keyboard in sophisticated ways.

- There is built-in support for laying out objects in *rows and columns*, for example, for menus, or in *graphs or trees*, for example, to show a directory structure or a dependency graph.

- Two complete sets of *gadgets* (also called widgets or interaction techniques) are provided to help the programmer get started. These include menus, buttons, scroll bars, sliders, circular gauges, graphic selection, scrollable windows, and arrows. More gadgets are being added all the time. One set has the Garnet look and feel, and the other has the Motif look and feel. The Motif set is implemented entirely in Lisp on top of Garnet, to provide maximum flexibility.

- Garnet is designed to be *efficient*. Even though Garnet handles many aspects of the interface automatically, an important goal is that it execute quickly and not take too much memory. We are always working to improve the efficiency, but Garnet can currently handle dozens of constraints attached to objects that are being dragged with the mouse.

- Garnet will automatically produce Postscript for any picture on the screen, so the programmer does not have to worry about printing.

- Gesture recognition (such as drawing an "X" over an object to delete it) is supported, so designers can explore innovative user interface concepts.

4. Coverage

Garnet is designed to handle interfaces containing a number of graphical objects which the user can manipulate with the mouse and keyboard.

Garnet is suitable for applications of the following kinds:

- User interfaces for expert systems and other AI applications.

- Box and arrow diagram editors like Apple Macintosh MacProject (which helps with project management).

- Graphical Programming Languages where computer programs can be constructed using icons and other pictures (a common example is a flowchart).
- Tree and graph editing programs, including editors for semantic networks, neural networks, and state transition diagrams.
- Conventional drawing programs such as Apple Macintosh MacDraw.
- Simulation and process monitoring programs where the user interface shows the status of the simulation or process being monitored, and allows the user to manipulate it.
- User interface construction tools (Garnet was implemented using itself).
- Some forms of CAD/CAM programs.
- Icon manipulation programs like the Macintosh Finder (which allows users to manipulate files).
- Board game user interfaces, such as Chess.

Figure 1 shows a simple Garnet application that was created from start to finish (including debugging) in three hours. The code for this application is shown in the sample program manual, which begins on page 501.

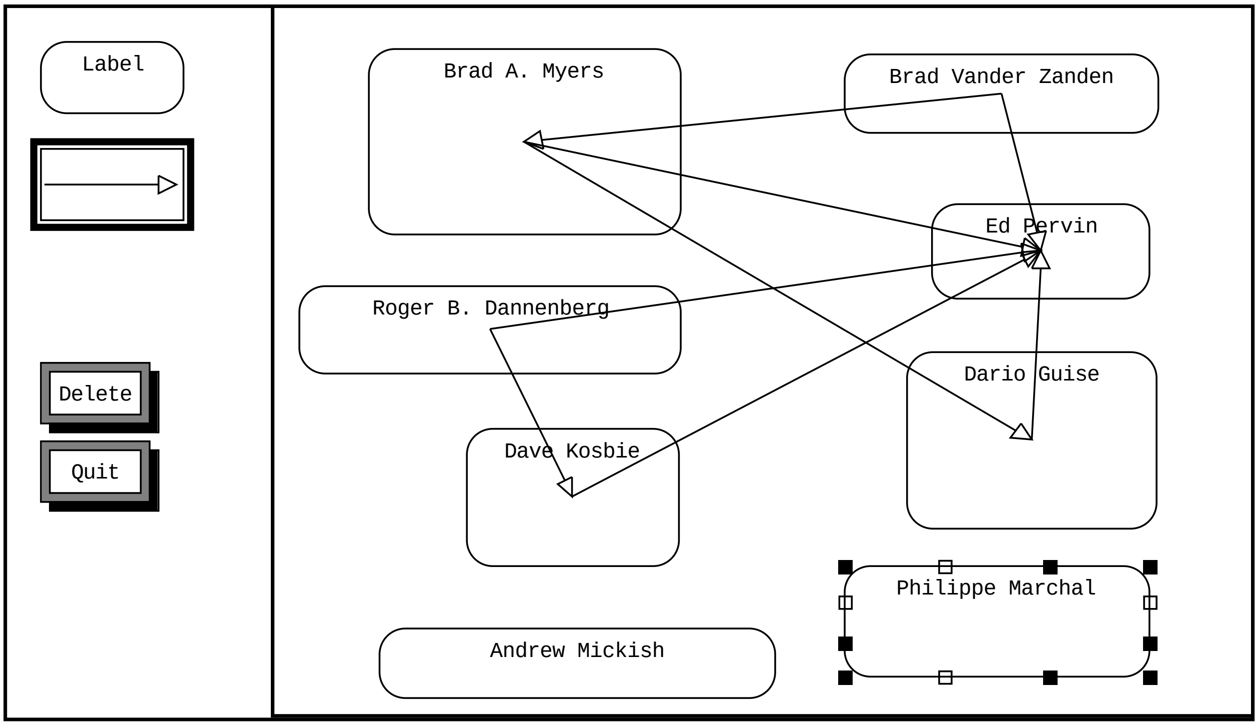

Figure 1: A sample Garnet application. The code for this application is listed the "Sample Garnet Program" section of this manual, starting on page 501.

Other examples of applications created using Garnet appear in the picture section of this manual, starting on page 23.

5. Running Garnet From /afs

If you are running Garnet from CMU, or if you have access to AFS, you can access Garnet directly on the `/afs` servers. The official release of Garnet is stored in

```
/afs/cs.cmu.edu/project/garnet/garnet/src/
/afs/cs.cmu.edu/project/garnet/garnet/lib/
/afs/cs.cmu.edu/project/garnet/garnet/doc/
```

We maintain binaries of the official release version in machine- and lisp-specific subdirectories of `/afs/cs/project/garnet/`. If you are at CMU, you can skip section 6 altogether, and just start lisp and load Garnet with:

```
(load "/afs/cs/project/garnet/garnet-loader.lisp")
```

The CMU version of `garnet-loader.lisp` will attempt to determine what kind of machine and lisp you're using, and load the appropriate binaries for you. You will not have to supply or customize any pathnames.

6. How to Retrieve and Install Garnet

Garnet is available for free by anonymous FTP.

Garnet uses the CLX interface from Lisp to X/11. CLX should be supplied with every Lisp. If you need help with CLX, you need to contact your Lisp vendor. We cannot help you acquire, compile, or install CLX, sorry.

The following instructions assume you run Unix and can FTP from CMU.

Retrieving the TAR Files:

The Garnet software is about 9 megabytes. In order to make it easy to copy the files over, we have created TAR files, so to use the mechanism below requires double the storage area. Therefore, you first need to find a machine with enough room, and then create a directory called `garnet` wherever you want the system to be:

```
% mkdir garnet
```

If you already have a copy of Garnet, you might want to move the old contents of the garnet directory somewhere, in case you need to go back. Otherwise, delete all the contents of the `garnet` directory.

Then, cd to the garnet directory.

```
% cd garnet
```

Now, ftp to `a.gp.cs.cmu.edu (128.2.242.7)`. When asked to log in, use `anonymous`, and your name as the password.

```
% ftp a.gp.cs.cmu.edu
Connected to A.GP.CS.CMU.EDU.
220 A.GP.CS.CMU.EDU FTP server (Version 4.105 of 10-Jul-90 12:07) ready.
Name (a.gp.cs.cmu.edu:bam): anonymous
331 Guest login ok, send username@node as password.
Password:
230 Filenames can not have '/..' in them.
```

Then change to the `garnet` directory (note the double `garnet`'s) and use binary transfer mode:

```
ftp> cd /usr/garnet/garnet/
ftp> bin
```

The files have all been combined into TAR format files for your convenience. These will create the appropriate sub-directories automatically. We have both compressed and uncompressed versions. For the regular versions, do the following:

```
ftp> get src.tar
ftp> get lib.tar
ftp> get doc.tar
```

To get the compressed version, do the following:

```
ftp> get src.tar.Z
ftp> get lib.tar.Z
ftp> get doc.tar.Z
```

Now you can quit FTP:

```
ftp> quit
```

Installing the Source Files:

If you got the compressed versions, you will need to uncompress them:

```
% uncompress src.tar.Z
% uncompress lib.tar.Z
% uncompress doc.tar.Z
```

Now, for each tar file, you will need to "untar" it, to get all the original files:

```
% tar -xvf src.tar
% tar -xvf lib.tar
% tar -xvf doc.tar
```

This will create subdirectories will all the sources in them. At this point you can delete the original tar files, which will free up a lot of space:

```
% rm *.tar
```

Now, copy the files `garnet-loader.lisp`, `garnet-compiler.lisp`, `garnet-prepare-compile.lisp`, and `garnet-after-compile` from the src directory into the `garnet` directory:

```
% cp src/garnet-*  .
```

Customizing the PathNames:

The file `garnet-loader.lisp` contains path names for all the parts of Garnet. You will now need to edit `garnet-loader.lisp` in an editor, and set up path names for all the parts of Garnet. Comments in the file will direct you how to do this. At the top of the file are two variables you will need to set: `Your-Garnet-Pathname` and `Your-CLX-Pathname`. These are used for all the `:external` branches of the loader.

```
% emacs garnet-loader.lisp   (or whatever editor you use)
```

Compiling Garnet to Make Binary Files:

Lisp requires very large address spaces. We have found on many Unix systems, that you need to expand the area that it is willing to give to a process. The following commands work in many systems. Type these commands to the C shell (csh). You might want to also put these commands into your `.login` file.

```
% unlimit datasize
% unlimit stacksize
```

Now, you will need to compile the Garnet source to make your own binaries. This is achieved by loading the compiler scripts. There is more information on compiling in section 9 below.

```
;; Only LispWorks users need to do the next two commands.  See section 8.7.
lisp> #+lispworks (load "src/utils/lispworks-processes.lisp")
lisp> #+lispworks (guarantee-processes)

lisp> (load "garnet-prepare-compile")
lisp> (load "garnet-loader")
lisp> (load "garnet-compiler")
```

Now Garnet is all compiled and loaded, so you can run Garnet code. To set up for the next time, however, it is best to quit lisp now, and run a shell script to move all the binaries to the correct places. If your sources are not in a directory named `garnet/src` or your binaries should not be in a directory named `garnet/bin`, then you will need to edit `garnet-after-compile` to set the directories. Also, if your compiler produces binary files that do not have one of the following extensions, then you need to edit the variable `CompilerExtension` in `garnet-after-compile`: ".fasl", ".lbin", or ".sbin". Otherwise, you can just execute the file as it is supplied (note: this is run from the shell, not from Lisp). You should be in the garnet directory.

```
% csh garnet-after-compile
```

Now you can start lisp again, and load Garnet:

```
lisp> (load "garnet-loader")
```

Details about how to customize the loading of garnet are provided in section 10.

7. Directory Organization

All of the information about where various files of Garnet are stored is in the file `garnet-loader.lisp`. This file also defines the Garnet version number:

```
* user::Garnet-Version-Number
"2.2"
```

You may need to edit the `garnet-loader` file to tell Garnet where all the files are. Normally, there will be a directory called `garnet` with sub-directories called `src`, `lib` and `bin`. In the `src` and `bin` directions will be sub-directories for all the parts of the Garnet system:

- `utils` - Utility files and functions.
- `kr` - KR object system.
- `opal` - Opal Graphics system.
- `inter` - Interactors input handling.
- `aggregadgets` - Files to handle aggregates and lists.
- `gadgets` - Pre-defined gadgets, such as menus and scroll bars.
- `gesture` - Tools for handling gestures as input.
- `ps` - Functions for printing Garnet windows with PostScript.
- `debug` - Debugging tools.
- `demos` - Demonstration programs written using Garnet.
- `gilt` - The Gilt interface builder.
- `c32` - A spreadsheet for editing constraints among objects.
- `lapidary` - The Lapidary interactive tool.
- `contrib` - Files contributed by Garnet users that are not supported by the Garnet group, but just provided for your use.

8. Site-Specific Changes

If you are transferring Garnet to your site, you will need to make a number of edits to files in order for Garnet to load, compile and operate correctly. All users will need to edit the Garnet pathnames as discussed in section 8.1, but relatively few users should need the other sections 8.2 - 8.8. Garnet has been adjusted to load on the widest possible variety of lisps and operating systems with minimum modification.

Of course, if you change any `.lisp` files in the Garnet subdirectories (not including `garnet-loader.lisp`), you will need to recompile them (section 11), even if you do not need to recompile other parts of Garnet.

8.1. Pathnames

After you have copied Garnet to your machine and untar'ed the source files, the top level Garnet directory will contain the file `garnet-loader.lisp`. This one file contains the file names for all the parts of Garnet. You should edit this file to put in your own file names. The best way to do this is to set the `Garnet-Version` to be `:external` and edit the string at the top of the file called `Your-Garnet-Pathname` to say where the files are. This change is normally done during the compile procedure, already described in section 6.

8.2. Compiler Optimization Settings

The variable `user::*default-garnet-proclaim*`, defined in `garnet-loader.lisp`, holds a list of compiler optimization flags and default values. These flags determine things like the size and speed of your resulting Garnet binaries. For example, the default value of this variable in Allegro is:

```
'(optimize (speed 3) (safety 1) (space 0) (debug 3))
```

This optimization causes Allegro to generate compiled binaries that are as fast and small as possible. The *safety* setting of 1 means that the compiled code will allow keyboard interrupts if you somehow go into an infinite loop, and the *debug* setting of 3 means you will get the most helpful error messages that Allegro can give you when you are thrown into the debugger.

Different implementations of lisp require different values for the optimization flags, and `garnet-loader.lisp` provides values for Allegro, Lucid, CMUCL, and LispWorks that we have found work particularly well. You can override the default optimizations by defining the `*default-garnet-proclaim*` variable before loading `garnet-loader.lisp`. A value of NIL for this variable means that you want to maintain the declarations that are already in effect for your lisp.

8.3. Fonts

In X11/R4 and R5, there are almost always a full set of fonts available with standard names. Garnet relies on these fonts being available on the standard font paths set up by X/11. You can try loading Garnet and see if it finds the standard fonts.

If not, look in the file `garnet/src/opal/text-fonts.lisp`. This file constructs font names according to the standard X/11 format (with lots of "-*-*-*"'s). You will have to substitute the names of fonts that are available at your installation.

8.4. Keyboard Keys

If your keyboard has some specially-labeled keys on it, Garnet will allow you to use these as part of the user interface. The file

```
define-keys.lisp
```

which is in the `garnet/src/inter` sub-directory, defines the mappings from the codes that come back from X/11 to the special Lisp characters or atoms that define the keys in Garnet.

For many machines, such as IBM RT, Sun, and DECStations, we have built in mappings for all of the keyboard keys. Since there are no Lisp characters for the special keys, they are named with keywords such as `:uparrow` and `:F1`. If some keys on your keyboard are not mapped to keywords, you can use the following mechanism to set this up.

To find the correct codes to use for each undefined key, load the file

```
find-key-symbols.lisp
```

which is in `garnet/src/inter/`. After loading this file, simply type the keys you need to find mappings for. It will print out the code number.

Then, you can go into the file `define-keys.lisp` and edit it so the codes you found map to appropriate keywords.

Next, you might want to bind these keys to keyboard editing operations. If you want these to be global to all Garnet applications, then you can edit the files

```
textkeyhandling.lisp
```

and

```
multifont-textinter.lisp
```

which contain the default mappings of keyboard keys to text editing operations. The Interactors Manual contains full more information on how this works.

If you surround your changes to all these files with `#+<your-switch>` and mail them back to us (`garnet@cs.cmu.edu`), then we will incorporate them into future versions so you won't need to continually edit the files.

8.5. Multiple Screens

If you are working on a machine with only one screen, you need not pay attention to this section. However, certain machines, such as the color Sun 3/60, have more than one screen. The color Sun 3/60 has both a black-and-white screen (whose display name is "unix:0.0") and a color screen (whose display name is "unix:0.1"). If you type "echo $DISPLAY" in a Unix shell, you will get the display name of the screen you are working on; that name should look like "unix:0.*" where * is some integer.

Garnet assumes that the DISPLAY environment variable has this form of "displayname:displaynumber.screennumber", and extracts the display and screen numbers from that. If any fields are missing, then the missing display or screen number defaults to zero.

8.6. OpenWindows Window Manager

If you are running OpenWindows from Sun, you will need to add the following line to your `.Xdefaults` file to make text input work correctly:

```
OpenWindows.FocusLenience:          True
```

8.7. LispWorks

LispWorks is the common lisp sold by Harlequin Ltd. There is one peculiarity about LispWorks that requires an additional step before executing the main-event-loop background process of Garnet (Garnet uses multiprocessing by default in LispWorks -- see the Interactors Manual, section "The Main Event Loop" for details). You need to perform this step both when <u>compiling</u> and <u>loading</u> Garnet (the appropriate steps are mentioned during the standard compile procedure in section 6).

LispWorks has an unconventional "initialization phaze" to multiprocessing, which requires that a special function be called before launching a background process. There are two ways to initialize multiprocessing in LispWorks. One way is to start the big window-oriented LispWorks interface by executing `(tools:start-lispworks)`. This will cause a menu to appear, and you can open a lisp listener as a selection from the menu. From this listener, you can load `garnet-loader.lisp`, and Garnet's main-event-loop process will be launched by default.

If you do not need all the functionality of the LispWorks interface, you can initialize multiprocessing with much less overhead. Before loading Garnet, load the file `"src/utils/lispworks-process.lisp"` and execute the function `guarantee-processes` to start multiprocessing. For example, at the LispWorks prompt you could type:

```
> (garnet-load "utils-src:lispworks-process.lisp")
> (guarantee-processes)

;; At this point, a new lisp listener has been spawned
> (load "garnet/garnet-loader")
```

It is important to realize that when you call `guarantee-processes`, a new lisp listener is spawned, and all subsequent commands will be typed into the second listener. Putting the `guarantee-processes` call at the top of the `garnet-loader.lisp` file will not work, because the first listener will remain hung at the `guarantee-processes` call, while the second process is waiting for user input.

Whenever you enter the debugger of the new listener spawned by `guarantee-processes`, you will get restart options that include:

```
...
5 (abort) return to level 0.
6 Return to top level
7 Return from multiprocessing
```

When you want to exit the debugger, you should choose either "(abort) return to level 0," or "Return to top level", since both of these options will return you to the top-level LispWorks prompt. If you ever choose "Return from multiprocessing", then you will kill both the second listener and the main-event-loop-process, and you will have to call `guarantee-processes` and `opal:launch-main-event-loop-process` to restart Garnet's main-event-loop process.

It is not necessary to load `"lispworks-process.lisp"` or execute `guarantee-processes` if you instead choose to execute `tools:start-lispworks`.

8.8. CLISP

CLISP is a Common Lisp (CLtL1) implementation by Bruno Haible of Karlsruhe University and Michael Stoll of Munich University, both in Germany. There are a couple of additional steps you must take to run Garnet in CLISP that are not required in other lisps.

Renaming .lisp files to .lsp

Before starting the procedure to compile Garnet, you will have to rename all of the source files from ".lisp" to ".lsp". A `/bin/sh` shell script has been provided to automate this process in the file `src/utils/rename-for-clisp`. This script requires that you `cd` into the `src` directory and execute

```
% sh utils/rename-for-clisp
```

The script will rename all of the `"src/*/*.lisp"` files to `".lsp"`, so that they can be read by CLISP.

Obtaining CLX

If you are already using CLISP, you may need to additionally retrieve the CLX module. CLX for CLISP can be retrieved via `ftp` from `ma2s2.mathematik.uni-karlsruhe.de`, in the file `/pub/lisp/clisp/packages/pcl+clx.clisp.tar.z`.

Making a Garnet image

Once you have installed the CLX module, you can make a restartable image of Garnet with the following procedure (NOTE: this is different from other lisps). This is the standard procedure for compiling Garnet, followed by a dump of the lisp image.

```
clisp -M somewhere/clx.mem
> (load "garnet-prepare-compile.lsp")
> (load "garnet-loader.lsp")
> (load "garnet-compiler.lsp")
> (opal:make-image "garnet.mem" :quit t)
```

The saved image can then be restarted with the command:

```
clisp -M garnet.mem
```

8.9. AKCL

Some of the default parameters for the AKCL lisp image are insufficient for running Garnet. You may be able to change some of these parameters in the active lisp listener, but it is probably better to rebuild your AKCL image from scratch with the following parameter values:

```
MAXPAGES for AKCL should be at least 10240, and

(SYSTEM:ALLOCATE-RELOCATABLE-PAGES 800)
(SYSTEM:ALLOCATE-CONTIGUOUS-PAGES 45 T)
(SYSTEM:ALLOCATE 'CONS   3500 t)
(SYSTEM:ALLOCATE 'SYMBOL  450 t)
(SYSTEM:ALLOCATE 'VECTOR  150 t)
(SYSTEM:ALLOCATE 'SPICE   300 t)
(SYSTEM:ALLOCATE 'STRING  200 t)
```

Garnet runs about half as fast in AKCL as on other Common Lisps. Increasing the RAM in your machine may help. Users have reported that 16MB on a Linux-Box 486 yields unacceptable performance.

9. Compiling Garnet

After executing the compile procedure in section 6, the result should be that all the files are compiled and loaded. (If there was a problem and you need to restart the compile proceedure, please see below.) The compiler scripts do *not* check for compile errors; that is up to you. We have attempted to make Garnet compile without errors on all common lisps, but some lisps generate more warnings than others.

The compiler scripts compile the binaries into the same directories as the source files. For example, all the interactor binaries will be in `garnet/src/inter/` along with the source (`.lisp`) files. Therefore, after the compilation is completed, you will need to move the binaries into their own directory (e.g., `garnet/bin/inter`). To do this, use the c-shell script

```
csh garnet-after-compile
```

The `garnet-after-compile` file will normally be in the top level garnet directory. Note that this is typed to the shell, not to Lisp. Even if you normally run the "regular" (Bourne) shell (sh), the above command should work.

To prevent certain parts of Garnet from being compiled, set `user::compile-`*xxx*`-p` to NIL, where *xxx* is replaced with the part you do not want to compile. See the comments at the top of the file `garnet-prepare-compile` for more information.

If you ever have to restart the compile process, you do not have to start from scratch. If you have not yet moved the binary files out of the `src/` directory (i.e., you have not yet run `garnet-after-compile`), then you can use the files that have been compiled already instead of compiling them again. Restart lisp, and for each Garnet module that has been compiled, set the variable `user::compile-`*xxx*`-p` to NIL to indicate that it should not be compiled again. Then load the three script files again in the usual order. Note: if a module has been only partially compiled, then you must recompile the whole module.

10. Loading Garnet

To load Garnet, it is only necessary to load the file:

```
(load "garnet-loader")
```

(Of course, you may have to preface the file name with the directory path of where it is located. It is usually in the top level `garnet` directory.)

To prevent any of the Garnet sub-systems from being loaded, simply set the variable `user::load-`*xxx*`-p` to NIL, where *xxx* is replaced by whatever part you do not want to load. Normally, some parts of the system are not loaded, such as the gadgets and demos. This is because you normally do not want to load or use all of these in a session. Files that use gadgets will load the appropriate ones automatically, and the `demos-controller` program loads the demos as requested.

It is possible to save an image of lisp after loading Garnet, so that when you restart lisp, Garnet will already be loaded and you will not have to load `garnet-loader.lisp`. For details about making lisp images, see the function `opal:make-image` in the Opal manual.

11. Loader and Compiler Functions

11.1. Garnet-Load and Garnet-Compile

There are two functions that allow you to save a lot of typing when you load and compile files. When you supply `garnet-load` and `garnet-compile` with the Garnet subdirectory that you want to get a file from (e.g., "gadgets"), the functions will automatically append your Garnet pathname to the front of the specified file.

`user::Garnet-Load "`*prefix*`:`*filename*`"` [*Function*]

`user::Garnet-Compile "`*prefix*`:`*filename*`"` [*Function*]

These functions are defined in `garnet-loader.lisp` and are internal to the `user` package.

The *prefix* parameter corresponds to one of the Garnet subdirectories, and the *filename* is a file in that directory. A list of the most useful prefixes appear in section 11.2, and a full list can be seen by evaluating the variable `user::Garnet-Load-Alist` in your lisp (after loading Garnet). Examples:

```
* (garnet-load "gadgets:v-scroll-loader")
Loading #p"/afs/cs/project/garnet/bin/gadgets/v-scroll-loader"
Loading V-Scroll-Bar...
...Done V-Scroll-Bar.

T
* (garnet-compile "opal:aggregates")
Compiling #p"/afs/cs/project/garnet/src/opal/aggregates.lisp"
for output to #p"/afs/cs/project/garnet/bin/opal/aggregates.fasl"
...
; Writing fasl file "/afs/cs/project/garnet/bin/opal/aggregates.fasl"
; Fasl write complete

NIL
*
```

There are two groups of prefixes that `garnet-load` accepts -- the "bin" prefixes and the "src" prefixes. `Garnet-load` assumes that when you load files, you will want to load the compiled binaries. Therefore, when you use prefixes like "gadgets", `garnet-load` uses the Garnet-Gadgets-Pathname variable to find the file you want. If you really want to load a file from your source directory, you should use the subdirectory name with "-src" tacked on. For example,

```
* (garnet-load "gadgets-src:motif-parts")
Loading #p"/afs/cs/project/garnet/src/gadgets/motif-parts"
...

T
*
```

`Garnet-compile` does not accept "-src" prefixes, because it always assumes that you want to take a lisp file from your source directory, compile it, and output it to your bin directory. Note: do not specify ".lisp" or ".fasl" with your filename -- `garnet-compile` will supply suffixes for you.

`Garnet-compile` attempts to determine your correct binary extension (".fasl", ".lbin", etc.) from the kind of Lisp that you are using. If `garnet-compile` ever gets the extension wrong, you can change it by setting the variable `*compiler-extension*`, which is defined in the `user` package.

11.2. Adding Your Own Pathnames

The functions `user::garnet-load` and `user::garnet-compile` look up their prefix parameters in an association list called `user::Garnet-Load-Alist`. Its structure looks like:

```
(defparameter Garnet-Load-Alist
  '(("opal" . Garnet-Opal-Pathname)          ; For loading the  multifont-loader
    ("gg" . Garnet-Gadgets-PathName)         ; For loading gadgets
    ("gestures" . Garnet-Gestures-PathName)  ; For loading  agate
    ("demos" . Garnet-Demos-PathName)        ; For loading demos
    ("gilt" . Garnet-Gilt-PathName)          ; For loading high-level tools...
    ("c32" . Garnet-C32-PathName)
    ("lapidary" . Garnet-Lapidary-PathName)
    ...))
```

This alist is expandable so that you can include your own prefixes and pathnames. Prefixes can be added with the following function:

`user::Add-Garnet-Load-Prefix` *prefix pathname* [*Function*]

For example, after executing `(add-garnet-load-prefix "home" "/usr/amickish/")`, you would be able to do `(garnet-load "home:my-file")`.

12. Overview of the Parts of Garnet

Garnet is composed of a number of sub-systems, some of which can be loaded and used separately from the others. Most of the subsystems also have their own separate packages. The following list shows the components of Garnet, the package used by that component, and the page number of the reference manual for that part in this technical report.

- `KR` - Package `kr`. The object and constraint system. Page 99.
- `Opal` - Package `opal`. The graphical object system. Page 151.
- `Interactors` - Package `inter`. Handling of mouse and keyboard input. Page 221.
- `Gestures` - Package `inter`. Code to handle gesture recognition and training. Described in the interactors manual, page 221.
- `Aggregadgets` - Package `opal`. Support for creating instances of collections of objects, and for rows or columns of objects. Page 299.
- `AggreGraphs` - Package `opal`. Support for creating graphs and trees of objects. Also described in the aggregadgets manual, page 299.
- `Gadgets` - Package `garnet-gadgets`, nicknamed `gg`. A collection of pre-defined gadgets, including menus, buttons, scroll bars, circular gauges, graphics selection, etc. Page 353.
- `Debugging tools` - Package `garnet-debug`, nicknamed `gd`. Useful functions to help debug Garnet programs, including the Inspector. Page 469.
- `Demonstration programs` - Each demonstration program is in its own package. Page 489.
- `Gilt` - Package `gilt`. The Garnet interface builder. Page 513.
- `C32` - Package `c32`. A spreadsheet interface for editing constraints. Page 533.
- `Lapidary` - Package `Lapidary`. A sophisticated interactive design tool. Page 545.

- `Contrib` - A set of file contributed by Garnet users. These have not been tested by the Garnet group, and are not supported. Each file should have a comment at the top describing how it works and who to contact for help and more information.

13. Overview of this Technical Report

In addition to the reference manuals for all the parts of the Garnet toolkit listed above, this technical report also contains:

- A guided on-line tour of the Garnet system that will help you become familiar with a few of the features of the Garnet toolkit. Page 37.
- A tutorial to teach you the basic things you need to know to use Garnet. Page 57.
- The code for a simple graphical editor, as a sample of code written for Garnet. Page 499.
- Some helpful hints that have been collected from the experience of Garnet users for making Garnet programs run faster. Page 581. If you have ideas for things to add to this section, let us know.

14. What You Need To Know

Although this is a large technical report, you certainly do not need to know everything in it to use Garnet. Garnet is designed to support many different styles of interface. Therefore, there are many options and functions that you will probably not need to use.

In fact, to run the Tour (page 37), it is not necessary to read any of the reference manuals. The tour is self-explanatory.

Next, you should probably read the Tutorial (page 57), since it tries to provide enough information about most of Garnet so that you don't need the other manuals right away.

To run the Gilt Interface Builder, you do not need to know about the rest of the system either. The Gilt manual should be sufficient. When you are ready to set some properties of the gadgets, you will need to look up the particular gadget in the Gadgets manual to see what the properties do.

Even when you are ready to start programming, you will still not need most of the information described here. To start, you should probably do the following:

1. Read this overview.
2. Run the tour, to get a feel for Garnet programming.
3. Read the tutorial.
4. You might try creating a few dialog boxes using Gilt. This will familiarize you with the Gadgets. See the Gilt manual (Page 513).
5. After that, you can look at the sample program at the end of this technical report, to see what you need more information about.
6. You could now try to start writing your own programs, and just use the rest of the manuals as reference when you need information.
7. Next, look at the introduction and the following functions in the KR document: `gv`, `gvl`, `s-value`, `formula`, `o-formula`, and `create-instance`. The KR manual documents the entire KR module, but Garnet does not use every feature that KR provides. Some concepts (like demons), will never be used by the typical Garnet user. Once you have gained some familiarity with the system, you may want to return to the KR Manual and read

about object-oriented programming, type-checking, and constants.

8. Next, skim the first five chapters of the Opal manual, and look at the various graphical objects, so you know what kinds are provided. The primary functions you will use from Opal are: `add-component`, `update`, and `destroy`, as well as the various types of graphical objects (`rectangle`, `line`, `circle`, etc.), drawing styles (`thin-line`, `dotted-line`, `light-gray-fill`, etc.) and fonts.

9. Next, in the Interactors manual, you will need to skim the first four chapters to see how interactors work, and then see which interactors there are in the next chapter. You will probably not need to take advantage of the full power provided by the interactors system.

10. Aggregadgets and Aggrelists are very useful for handling collections of objects, so you should read their manual. They support creating instances of groups of objects.

11. You should then look at the gadget manual to see all the built-in components, so you do not have to re-invent what is already supplied.

12. User interface code is often difficult to debug, so we have provided a number of helpful tools. The Inspector is mentioned briefly in the Tutorial, and it is discussed thoroughly in the debugging manual. You will probably find many debugging features very useful.

13. The demo programs can be a good source of ideas and coding style, so the document describing them might be useful.

If all you want Garnet for is to display menus and gauges that are supplied in the gadget set, you can probably just read the KR, Gadgets and Gilt manuals, and skip the rest.

15. Garnet Articles

A number of articles about Garnet have been made available for FTP from the directory `/usr/garnet/garnet/doc/papers/` on `a.gp.cs.cmu.edu`. There is a README file in that directory, indicating which `.ps` files correspond to the Garnet bibliography citations.

The following is a complete list of articles written about Garnet, as of September 3, 1993. There are 32 refereed journal articles, 3 book chapters, 13 tech reports, and 5 articles submitted for publication.

Refereed Journal Articles

Robert J.K. Jacob, John J. Leggett, Brad A. Myers, and Randy Pausch. ‘‘Interaction Styles and Input/Output Devices,’’ *Behaviour and Information Technology*. March-April, 1993. Vol. 12, no. 2. pp. 69-79.

Dan R. Olsen, Jr., James D. Foley, Scott E. Hudson, James Miller, and Brad Myers. ‘‘Research Directions for User Interface Software Tools,’’ *Behaviour and Information Technology*. March-April, 1993. Vol. 12, no. 2. pp. 80-97.

Brad A. Myers and Brad Vander Zanden. ‘‘Environment for Rapid Creation of Interactive Design Tools,’’ *The Visual Computer; International Journal of Computer Graphics*. Vol 8, No. 2, February, 1992. pp. 94-116.

Brad A. Myers, Dario A. Giuse, Roger B. Dannenberg, Brad Vander Zanden, David S. Kosbie, Ed Pervin, Andrew Mickish, and Philippe Marchal. ‘‘Garnet; Comprehensive Support for Graphical, Highly-Interactive User Interfaces,’’ *IEEE Computer*. Vol. 23, No. 11. November, 1990. pp. 71-85. Translated into Japanese and reprinted in *Nikkei Electronics*, No. 522, March 18, 1991, pp. 187-205.

Brad A. Myers. ‘‘A New Model for Handling Input,’’ *ACM Transactions on Information Systems*. Vol. 8, No. 3. July, 1990. pp. 289-320.

Refereed Conference Articles

Brad A. Myers, Richard G. McDaniel, and David S. Kosbie. ‘‘Marquise: Creating Complete User Interfaces by Demonstration,’’ *Proceedings INTERCHI'93: Human Factors in Computing Systems*. Amsterdam, Holland, April 24-29, 1993. pp. 293-300.

Brad A. Myers, Richard Wolf, Kathy Potosnak, and Chris Graham. ‘‘Heuristics in Real User Interfaces,’’ *Proceedings INTERCHI'93: Human Factors in Computing Systems*. Amsterdam, Holland, April 24-29, 1993. pp. 304-307.

James A. Landay and Brad A. Myers. ‘‘Extending an Existing User Interface Toolkit to Support Gesture Recognition.’’ *Adjunct Proceedings of INTERCHI'93*. Amsterdam, Holland, April 24-29, 1993. pp. 91-92.

Brad A. Myers. ‘‘The Garnet Gilt Interface Builder: Graphical Styles and Tabs and Techniques for Reducing Call-Back Procedures,’’ Application Builder Session, *Seventh Annual X Technical Conference,* Boston, Massachusetts, January 18, 1993.

Osamu Hashimoto and Brad A. Myers. ‘‘Graphical Styles For Building User Interfaces by Demonstration,’’ *ACM Symposium on User Interface Software and Technology*, Monterey, CA, Nov. 16-18, 1992. pp. 117-124.

Brad A. Myers, Dario A. Giuse, and Brad Vander Zanden. ‘‘Declarative Programming in a Prototype-Instance System: Object-Oriented Programming Without Writing Methods,’’ *Proceedings OOPSLA'92: ACM Conference on Object-Oriented Programming Systems, Languages, and Applications*. October 18-22, 1992. Vancouver, BC, Canada. *SIGPLAN Notices,* vol. 27, no. 10. pp. 184-200.

Brad A. Myers and Mary Beth Rosson. ‘‘Survey on User Interface Programming,’’ *Proceedings SIGCHI'92: Human Factors in Computing Systems*. Monterrey, CA, May 3-7, 1992. 195-202.

Brad A. Myers. ‘‘Separating Application Code from Toolkits: Eliminating the Spaghetti of Call-Backs,’’ *ACM Symposium on User Interface Software and Technology*, Hilton Head, SC, Nov. 11-13, 1991. pp. 211-220.

Brad Vander Zanden, Brad A. Myers, Dario Giuse and Pedro Szekely. ‘‘The Importance of Indirect References in Constraint Models,’’ *ACM Symposium on User Interface Software and Technology*, Hilton Head, SC, Nov. 11-13, 1991. pp. 155-164.

Brad A. Myers. ‘‘Graphical Techniques in a Spreadsheet for Specifying User Interfaces,’’ *Proceedings SIGCHI'91: Human Factors in Computing Systems*. New Orleans, LA. April 28-May 2, 1991. pp. 243-249.

Brad Vander Zanden and Brad A. Myers, ‘‘Automatic, Look-and-Feel Independent Dialog Creation for Graphical User Interfaces,’’ *Proceedings SIGCHI '90: Human Factors in Computing Systems*. Seattle, WA, April 1-5, 1990. pp. 27-34.

Brad A. Myers. ‘‘Making it Easy to Create Highly-Interactive, Graphical Applications in Lisp,’’ *XNextEvent: The Official Newsletter of XUG, The X User's Group*. Vol. 3., No. 1. April, 1990.

Brad A. Myers. ‘‘An Object-Oriented, Constraint-Based, User Interface Development Environment for X in CommonLisp,’’ *Fourth Annual X Technical Conference*, Boston, Massachusetts, January 15-17, 1990.

Dario Giuse. ‘‘Efficient Frame Systems,’’ *Lecture Notes in Artificial Intelligence, EPIA 1989*. Vol. 390. J.P. Martins and E.M. Morgado, eds. Springer-Verlag, Sep, 1989.

Brad A. Myers, Brad Vander Zanden, and Roger B. Dannenberg. ‘‘Creating Graphical Interactive Application Objects by Demonstration,’’ *ACM Symposium on User Interface Software and Technology*, Williamsburg, VA, Nov. 13-15, 1989. pp. 95-104.

Charles Wiecha, Stephen Boies, Mark Green, Scott Hudson, and Brad Myers. ‘‘Direct Manipulation or Programming: How Should We Design Interfaces?’’ \fIACM Symposium on User Interface Software and Technology\fR, Williamsburg, VA, Nov. 13-15, 1989. pp. 124-126.

Brad A. Myers. ‘‘AI In Demonstrational User Interfaces,’’ *A New Generation of Intelligent Interfaces: IJCAI-89 Workshop*, Detroit, MI. August 22, 1989, pp. 84-91.

Dario Giuse. ‘‘Efficient Knowledge Representation Systems,’’ *The Knowledge Engineering Review*, Vol 4, no. 4, 1990.

Brad A. Myers. ‘‘Encapsulating Interactive Behaviors,’’ *Proceedings SIGCHI ’89: Human Factors in Computing Systems*. Austin, Texas, April 30 - May 4, 1989, pp. 319-324.

Brad A. Myers. ‘‘User Interface Tools: Introduction and Survey,’’ *IEEE Software*, Vol. 6, no. 1, Jan, 1989. pp. 15-23. To be reprinted in *Milestones in Software Evolution*, Paul W. Oman and Ted G. Lewis, ed. IEEE Computer Society Press.

Pedro Szekely and Brad Myers. ‘‘A User Interface Toolkit Based on Graphical Objects and Constraints,’’ *OOPSLA ’88: Conference on Object-Oriented Programming: Systems, Languages and Applications*, San Diego, CA, September 25-30, 1988. *Sigplan Notices*, Vol 23, no. 11, November, 1988. pp. 36 - 45.

Daniel Kuokka and Dario Giuse. ‘‘The Dante Application Interface,’’ *Proceedings of the 2nd IEEE International Conference on Computer Workstations*, Clara, California, March 7-10, 1988.

Dario Giuse. ‘‘Lisp as a Rapid Prototyping Environment: The Chinese Tutor.’’ In *Lisp and Symbolic Computation - An International Journal*, Kluwer Academic Publishers, May 1987.

Refereed Published Videotapes

Brad A. Myers, Andrew Mickish and Osamu Hashimoto. ‘‘The Garnet Gilt Interface Builder: Graphical Styles and Tabs and Techniques for Reducing Call-Back Procedures,’’ Application Builder Video Session, *Seventh Annual X Technical Conference*, Boston, Massachusetts, January 18, 1993. 10 minutes.

Brad Vander Zanden and Brad A. Myers. *Creating Graphical Interactive Application Objects by Demonstration: The Lapidary Interface Design Tool.* 12 minute videotape. Technical Video Program of the SIGCHI’91 conference, New Orleans, LA. April 28-May 2, 1991. *SIGGRAPH Video Review*, Issue 64, no. 1.

Brad A. Myers. *Some of the Widgets*. 17 minute videotape. Technical Video Program of Interact’90. Cambridge, England. August 27-31, 1990.

Brad A. Myers. *All the Widgets*. 2 hour videotape. Technical Video Program of the SIGCHI’90 conference, Seattle, WA. April 1-4, 1990. *SIGGRAPH Video Review*, Issue 57.

Chapters in Books

Brad A. Myers. ‘‘State of the Art in User Interface Software Tools,’’ *Advances in Human-Computer*

Interaction, Volume 4. Edited by H. Rex Hartson and Deborah Hix. Norwood, NJ: Ablex Publishing, 1993. pp. 110-150.

Brad A. Myers. ''Garnet: Uses of Demonstrational Techniques,'' *Watch What I Do: Programming by Demonstration*, Allen Cypher, et. al., eds. Cambridge, MA: The MIT Press, 1993. pp. 219-236.

Brad A. Myers. ''Ideas from Garnet for Future User Interface Programming Languages,'' *Languages for Developing User Interfaces*. Boston: Jones and Bartlett, 1992. pp. 147-157.

Technical Reports:

Brad A. Myers. *Why are Human-Computer Interfaces Difficult to Design and Implement?* Carnegie Mellon University School of Computer Science Technical Report, no. CMU-CS-93-183. July 1993.

Brad A. Myers, editor. *The Second Garnet Compendium: Collected Papers, 1990-1992*. Carnegie Mellon University School of Computer Science Technical Report, no. CMU-CS-93-108, February, 1993.

Brad A. Myers, Dario Giuse, Roger B. Dannenberg, Brad Vander Zanden, David Kosbie, Philippe Marchal, Ed Pervin, Andrew Mickish, James A. Landay, Richard McDaniel, Don Hopkins. *The Garnet Toolkit Reference Manuals: Support for Highly-Interactive, Graphical User Interfaces in Lisp*. Carnegie Mellon University Computer Science Department Technical Report, no. CMU-CS-90-117-R3, Nov. 1992. Revised from CMU-CS-90-117-R2, May 1992, CMU-CS-90-117-R, June 1991, CMU-CS-90-117, March, 1990, and CMU-CS-89-196, Nov. 1989.

Bonnie E. John, Philip L. Miller, Brad A. Myers, Christine M. Neuwirth, and Steven A. Shafer, eds. *Human-Computer Interaction in the School of Computer Science*. Carnegie Mellon University School of Computer Science Technical Report, no. CMU-CS-92-193, October, 1992.

Brad A. Myers. *State of the Art in User Interface Software Tools*. Carnegie Mellon University School of Computer Science Technical Report, no. CMU-CS-92-114, February, 1992.

Brad A. Myers and Mary Beth Rosson. *Survey on User Interface Programming*. Carnegie Mellon University School of Computer Science Technical Report, no. CMU-CS-92-113, February, 1992. Also published as IBM Research Report RC17624.

Brad A. Myers, editor. *The Garnet Compendium: Collected Papers, 1989-1990*. Carnegie Mellon University School of Computer Science Technical Report, no. CMU-CS-90-154, August, 1990.

David Kosbie, Brad Vander Zanden, Brad A. Myers, and Dario Giuse. ''Automatic Graphical Output Management,'' *The Garnet Compendium: Collected Papers, 1989-1990*, Carnegie Mellon University School of Computer Science Technical Report, no. CMU-CS-90-154, August, 1990, pp. 29-43.

Roger B. Dannenberg, Brad A. Myers, Dario Giuse, and David Kosbie. ''Using Aggregates as Prototypes,'' *The Garnet Compendium: Collected Papers, 1989-1990*, Carnegie Mellon University School of Computer Science Technical Report, no. CMU-CS-90-154, August, 1990, pp. 79-93.

Dario Giuse. *KR: Constraint-Based Knowledge Representation*. Carnegie Mellon University Computer Science Department Technical Report, no. CMU-CS-89-142, Apr, 1989.

Brad A. Myers. *The Garnet User Interface Development Environment; A Proposal*. Carnegie Mellon University Computer Science Department Technical Report, no. CMU-CS-88-153, Sept, 1988.

Brad A. Myers. *Tools for Creating User Interfaces: An Introduction and Survey,* Carnegie Mellon University Computer Science Department Technical Report, no. CMU-CS-88-107, Jan, 1988.

Dario Giuse. *KR: an Efficient Knowledge Representation System*. Carnegie Mellon University Robotics Institute Technical Report, no. CMU-RI-TR-87-23, Oct, 1987.

Submitted for Publication

Brad A. Myers. ‘‘User Interface Software Tools.’’

Brad A. Myers. ‘‘Why are Human-Computer Interfaces Difficult to Design and Implement?’’

Brad Vander Zanden and Brad A. Myers. ‘‘The Lapidary Graphical Interface Design Tool.’’

Brad Vander Zanden, Brad A. Myers, Dario Giuse and Pedro Szekely. ‘‘Integrating Pointer Variables into Constraint Models.’’

Brad A. Myers, Dario A. Giuse, Andrew Mickish, and David S. Kosbie. ‘‘Making Structured Graphics and Constraints Practical for Large-Scale Applications.’’

16. Planned Future Extensions

Garnet is under active development, and we have a number of significant extensions planned. Although our main future thrust will be on the higher-level tools, there are some significant improvements planned for the toolkit level:

- Support for multi-way (bi-directional) constraints.
- Ability to run using Macintosh QuickDraw under the Macintosh operating system.
- Marquise, a new high-level tool that allows all aspects of the user interface to be specified.

We would like your input as to which of these are most important and what other extensions are desired.

References

[Myers 90] Brad A. Myers, Dario A. Giuse, Roger B. Dannenberg, Brad Vander Zanden, David S. Kosbie, Edward Pervin, Andrew Mickish, and Philippe Marchal.
Garnet: Comprehensive Support for Graphical, Highly-Interactive User Interfaces.
IEEE Computer 23(11):71-85, November, 1990.

Index

Table of Contents