Using the Octopus

Francisco J. Ballesteros
nemo@lsub.org
5/7/2008

ABSTRACT

The Octopus is a distributed system built by centralizing everything on a central computer. It can be used to turn any machine on the Internet into a user's terminal to access the Octopus, that is, the central computer. Multiple machines may be used as terminals, simultaneously. The system includes a novel distributed window system and is able to exploit software and devices found at terminals, which may be running different systems. This paper is an introduction and a tutorial description to get users started with the system. It focuses on using Octopus with Plan 9 from Bell Labs as the central computer system, but most things said apply to other systems.

1. Introduction

In the Octopus [1], there is a **single dedicated computer per user**. We refer to it as *the computer*. If a user has more than one (which is the common case), one of them is designated as the computer. All user programs execute on the computer, irrespectively of the user's location and of the devices and resources required to run them. The central computer (or PC) may run the user's favorite operating system, Plan 9 from Bell Labs [4] in our case. Octopus is implemented on Inferno [2] and runs hosted on top of most operating systems without any problem.

All other machines are considered terminals for the PC. They are used to reach the PC and to provide resources for it. In this system model, the PC is not considered to have devices on its own. Devices, including mouse, keyboard, graphics, audio, and voice, are provided by terminals. The scheme is depicted in figure 1.

The central computer is assumed to be highly available. For Plan 9, this means that programs like `aux/reboot` should run on it to watch out for problems and reboot the machine if necessary. Also, it is suggested to employ a Plan 9 CPU server kernel, able to boot without requiring the user to type a password to bring the system into operation. The PC may be kept at a safe location should it be an issue.

Terminals on the other hand are considered to be volatile stateless machines. They may run any other operating system. The octopus software running on them provides a way to reach the central computer and also provides devices and resources from the terminal to the PC. Multiple terminals may be used at the same time by the same user to reach the PC. Also, some terminals may be left running at all times (and perhaps be left out unattended) while new ones from a different location might be employed.

From the user's perspective, the Octopus is a new interface for accessing the system running at the PC from any machine in the network, no matter its architecture or operating system. But this is not to say that it is a remote desktop software. It lets the user combine multiple machines and employ their devices at will.

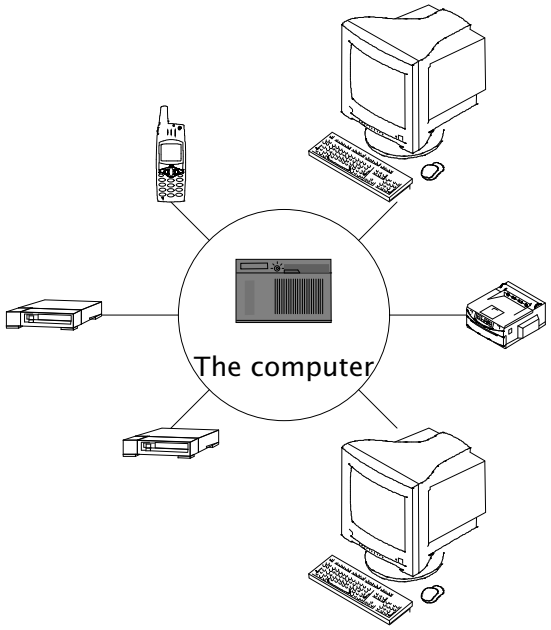

Figure 1: *System Model: The network is the bus to connect devices to the single computer*

2. System organization

Before describing how to use the system it is useful to learn how is it organized. The organization of the octopus is depicted in figure 2. It is important to recognize the four environments involved to avoid confusion. The PC runs Plan 9 from Bell Labs, and hosts an Inferno running the Octopus software for the PC. Two different terminals are attached to the PC in the figure, one running Plan 9 and another running MacOS X. Both terminals run a hosted Inferno running the Octopus software for a Terminal.

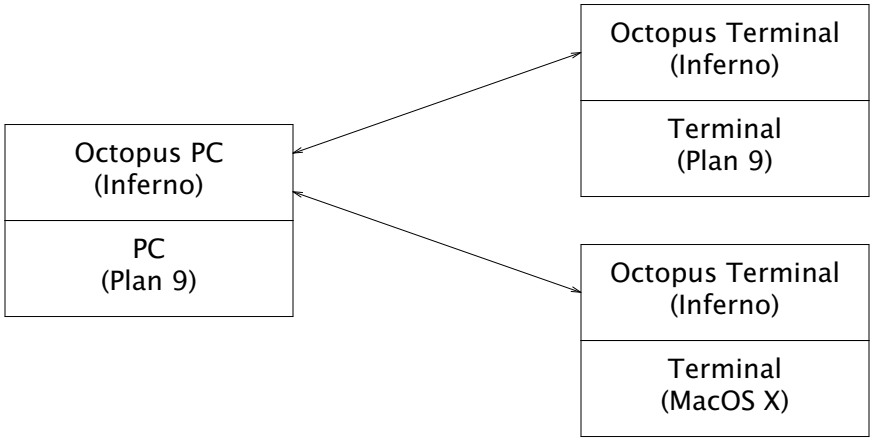

Figure 2: *Organization of the octopus*

Although most of the time the user would think he is using just the (Plan 9) PC, and he can forget about most of what we say here, it is important to know that there are four distinct environments:

1 The PC (running Plan 9 in the figure). User commands are executed here. Files browsed within the window system and used by commands come from the name space provided by this environment. On Plan 9 machines, all the name spaces from

terminals are also available within this environment.

2 The Inferno running at the PC. This is where the Octopus window system runs. This provides infrastructure to reach the underlying environment from terminals. Name spaces exported from terminals, providing their resources, are available at this environment.

3 The Inferno running at a terminal. This is where the window system viewer runs. The name space from the PC is available within this environment, as well as the local name space provided by the terminal native system.

4 The native system running at a terminal. (e.g., MacOS X in the down-right part of the figure). This provides applications used by the octopus as devices to view documents, browse the web, reproduce audio and voice, etc.

The user interacts directly with the terminal's Inferno environment, which executes the user interface for the window system and applications, called Olive. The *olive*(1) manual page documents this interface in detail.

By default, commands executed through Olive are executed at the PC (in the Plan 9 environment), but there are escapes to execute them at the Inferno hosted on the PC. Executing commands at the terminal is easy by opening a shell window on the Inferno running on it.

Files (and the name space) seen through Olive come also from the PC. However, it is easy to access files from the Inferno hosted on the PC because they are found at `/usr/inferno` on the PC environment. Regarding files from the terminal, they are always available through the local Inferno environment provided by the terminal.

2.1. PC name space

The (PC) name space where commands follows certain conventions documented here. Such conventions are the center of the Octopus and permit programs to add resources from terminals to the central computer.

The name space is a typical Plan 9 name space. This means that most conventions expected hold, like `/bin` for binaries, `/usr` for user files, etc. On installations using another system at the PC, the conventions from that system would be used instead.

The following files and directories are also available when using the PC from the Octopus:

- **/dev/snarf** is the clipboard. Usually it is an actual file at `$home/lib/snarf/snarf` bound to `/dev` along with the next file.
- **/dev/sel** contains the path for the (user interface) panel where a selection was last made. The file is kept at `$home/lib/snarf/sel` or a similar place and bound to `/dev` like before.
- **/mnt** contains directories where different resources added to the PC can be found.
- **/mnt/ui** contains the file system interface for the Octopus window system, Omero. See *omero*(4) for a description.
- **/mnt/ports** contains a directory providing the event delivery mechanism used by the Octopus.
- **/mnt/view** is a file viewer device. To view a file (e.g., a PDF) it suffices to copy it to this directory. A document viewer at a terminal near the user (probably one at the terminal being used to request the copy) will be launched as a result of the copy.
- **/mnt/view/ndb** is a text file describing properties of the view device. Among other things it describes the location of the device and the terminal providing it.
- **/mnt/print** is a printing device. To print a file, copy it to this directory. The file will be printed at the default printer queue provided by a terminal near the user.

- **/mnt/print/ndb** describes the print device. All devices include an `ndb` file by convention (although we will not mention it further).
- **/mnt/print/voice** is a voice synthesis device. To speak some text, copy it to the **speak** file it provides. As all other devices mounted at `/mnt` the device used will be one near the user.
- **/mnt/terms/***terminal* is a directory where devices and resources from the machine *terminal* can be found. There are multiple terminals mounted below `/mnt/terms` in the same way. We mention some devices next, but not all of them would be provided by all terminals. The devices found directly at `/mnt` come indeed from the ones we mention here.
- **/mnt/terms/***terminal***/view** is the view device provided by the terminal.
- **/mnt/terms/***terminal***/print** is the print device from that terminal.
- **/mnt/terms/***terminal***/voice** is its voice synthesis device.
- **/mnt/terms/***terminal***/fs** contains the file system from *sysname* (from its native OS, not from the Inferno it runs). This can be used to transfer files back and forth between the PC and the terminal.
- **/n/pc** contains the name space of the Inferno running at the PC. All resources described before in this list come from this file tree.
- **/what** contains information about machines.
- **/who** contains information about users.

2.2. PC's Inferno name space

The name space space at the Inferno running on the PC is similar to that of any other Inferno installation, but includes the following files and directories. Unless said otherwise, the directories mean the same the do in the name space of the PC.

- **/dis/o** contains the Dis binaries for the Octopus.
- **/dis/o/$emuhost** contains Dis binaries for the Octopus intended for the host system represented by the `$emuhost` variable. This directory is bound also at `/dis` to add platform dependent binaries to the portable ones.
- **/mnt/registry** is the mount point for the registry, describing resources known by the Octopus.
- **/mnt/ui**
- **/mnt/ports**
- **/mnt/snarf** contains the `snarf` and `sel` files found at `/dev` in the PC.
- **/mnt/view**
- **/mnt/print**
- **/mnt/voice**
- **/mnt/fs** contains the PC name space, including the files described before for the PC.
- **/mnt/what** is the PC `/what` directory, and contains information about machines.
- **/mnt/who** is the PC `/who` directory and contains information about users.
- **/terms** is similar to `/mnt/terms` in the PC.

2.3. Terminal's Inferno name space

The name space at the Inferno running on a terminal has the file tree of the PC's Inferno at `/pc` (using the Octopus protocol as the file protocol) and also at `/n/pc` (using Styx as the file protocol). The former works better on poor network connections but the later

is closer to expected semantics for file access. Also, /mnt/registry /mnt/snarf and other devices from the PC are available for use by terminal software.

2.4. Terminal name space

The terminal name space is not modified by the Octopus. However, as an aid for cases when it is desirable to execute native applications at the terminal that require Octopus services, the name space is still available.

On Plan 9 systems, /srv/pc can be used at the terminal to mount the file tree found at /pc at the terminal's Inferno. This corresponds to the name space at the PC containing most resources known to the Octopus.

On other systems, a WebDAV server executes at the terminal and mounting http://localhost:9999/pc achieves the same effect.

2.5. Environment information

The directory /who/$user contains information known about $user including a file named where with the last known location, a file named status with the user status, and a file named last with the name of the last terminal used by this user. Locations are strings without further structure, like 136 or home for example. By convention, the status may be online or busy when the user is available and away when the user is away from the computing environment. But note that a user connected from home is not considered to be away.

The directory /what/$sysname contains information about the $sysname system, including a file where with the system location.

When a terminal starts, it asks two things to the user: the address of the PC (where the Octopus software is listening) and the location of the terminal. This information is used to update /who and /what according to the activity of the user. It is important in practice because devices found at /mnt come from terminals colocated with the user, and location information is determined in this way.

3. Using the system

The first thing done to use the octopus is to start a terminal. How this is done depends on the native system used by the terminal. On MacOSX machines, downloading the bundle for a terminal and double clicking on the Octopus icon starts a terminal. On Plan 9 machines executing the octopus script suffices:

```
location? [home] press enter or type a location
PC? [nautilus.lsub.org] type enter or the address of the PC
welcome to your octopus terminal at home
PC is nautilus.lsub.org 192.147.71.68
importing PC
terminal with radius 209
  ...the window system starts...
```

To avoid typing, it is customary to keep the authentication ticket stored along with the rest of the terminal software. However, when borrowing a machine as a terminal, we might prefer to execute a different start script using getauthinfo to let us type our secret and keep it in memory before dialing the PC:

```
use signer [$SIGNER]: type enter
remote user name [nemo]: type enter
password: type your secret
save in file [yes]: press enter or type no
location? [home] type enter or type new location
 ...same as before...
```

This can be achieved by editing `/dis/o/termrc` in the Inferno distribution used at the terminal.

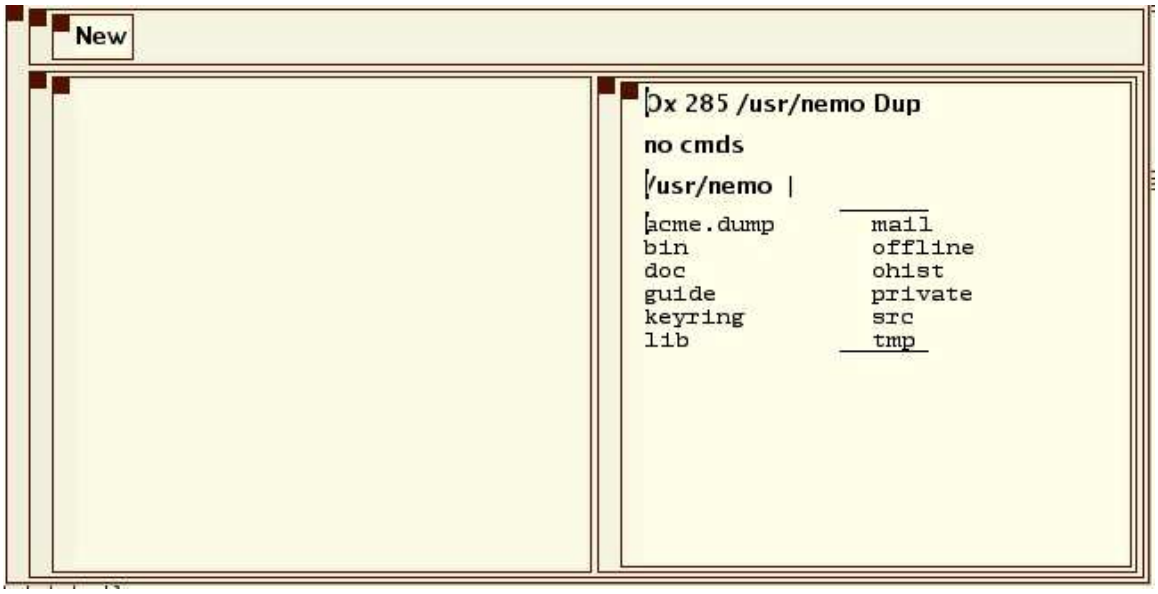

Figure 3: *The window system shown at the terminal.*

At this point the user may see a screen similar to the one shown in figure 3. The window sytem, Omero, is executing at the PC. A viewer, Olive, executes at the terminal and interfaces the window system to the user. Initially Omero contains a single screen like the one shown and Olive would attach to it without asking the user.

The Inferno executing Olive may be killed at any time without hesitation. Starting a new terminal would yield the same screen. Also, there is no need to exit a terminal before starting a new one (or new ones). Editions made at a terminal synchronize with the rest of the system upon mouse movement.

4. User interface

Each one of the rectangles shown in figure 3 is called a *panel*. The little brown box at the top-left corner of most panels is called a *tag*, and can be used to perform operations on the panel. For example, dragging the tag with the left mouse button (button-1) can be used to move a panel to another place.

The panel on the right is called a directory tree. Its first line shows that the program `Ox` with pid 285 is running on the PC, to let the user browse `/usr/nemo`. The second line shows that no commands have been started by the user. The third line contains the name for the directory shown, `/usr/nemo`.

We may open one of the files shown by, first, clicking on it with the right mouse button (button-3) to pop-up a menu and, second, moving the mouse quickly to the right in the direction of `Open`. Figure 4 shows the text menu used in this example. Options from the menu are selected by quick mouse movement toward the one desired. If you change your mind after calling a menu, a single click with button-1 disarms the menu without doing anything.

Files and directories open are shown in the same screen used to open them. Figure 5 shows the layout for the screen after opening the file `guide` and the directory `bin`.

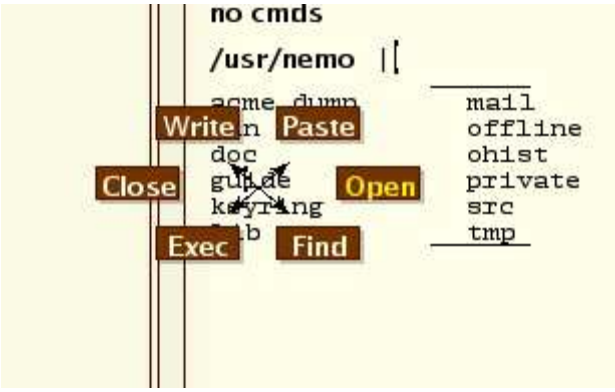

Figure 4: *The text menu. Use* Open *to open files and directories.*

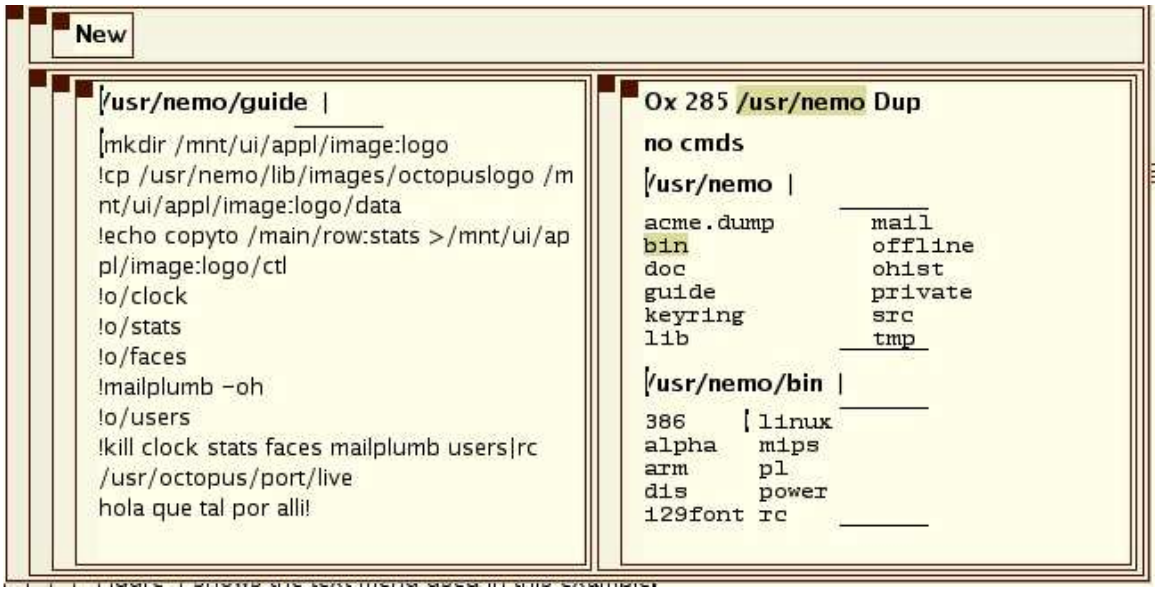

Figure 5: *After opening several files.*

All text panels show a little line marking the start and the end of the text. If the line is not shown, there is more text above or below the one shown. To scroll you may use the up and down arrow keys: ''↑'' and ''↓''. You may also raise the menu and then use button 1 to move up and down, and also use button 3 directly to scroll up or down by keeping it held while you move.

To close the file `guide` we may use the `Close` option from the menu on its panel. Along with each file shown there is a *tag line* shown for it (above it, in bold font). Its purpose is to inform the user about what is the text shown below it, and it is also a place to let you type some text (e.g., to execute commands).

We may also use the command language to control editions. For example, typing X and *Return* on a tag line would list the files being edited. The output may be similar to this, depending on the actual editions:

```
+  /usr/nemo
+  /usr/nemo/guide
+  /usr/nemo/bin
+  [/usr/nemo]
```

Each line corresponds to a file being shown. Panels with names between brackets correspond to output from commands.

Another way to execute a command is to select it (when consisting on several words) and use the menu option `Exec` on it.

Commands operate on the last panel selected by the user (where button 1 was used last time). Only file and directory text panels are considered in this respect. Clicking on a tag line does not alter the selected panel. The file `/dev/sel` reports the path for the panel, should you need it.

The command language is almost the same one used by the Sam editor [3], with a few changes. Thus, we might get rid of all editions of C source code by executing

```
X/*.[ch]/D
```

at any panel. And we may replace `foo` with `bar` at the entire selected panel by executing

```
, x/foo/c/bar
```

All commands are considered edition commands, unless escaped by a `!` sign. In that case, they are executed in the PC system using the directory of the selected panel as the current directory. For example, to view a document you may execute

```
!cp doc.pdf /mnt/view
```

to copy it to the view device near the user. To print the document at the default printer queue from terminal `nautilus` may be achieved in the same way:

```
!cp doc.pdf /mnt/terms/nautilus/print
```

Copying to the `fs` directory instead of copying to the `print` one may be used to copy the file to the terminal's file system.

Another escape, %, permits the execution of commands on the Inferno executing at the PC. This is most useful to recompile or update the Inferno software making up the Octopus. For example, executing

```
%mk clean ; mk install
```

at `/usr/octopus/port` would recompile and reinstall the octopus software. Note that execution of commands at the terminal's Inferno is simply a matter of starting a shell on that Inferno. Executing the command above in this way would update not the PC software but the one used by the terminal.

We can execute native commands in the host that use devices from the Octopus as you have seen. The screen shown in figure 6 corresponds to a system executing several graphical programs, like clock, stats, and faces.

4.1. Pitfalls

It is not clear if it is a user's pitfall or a design one, but it is important to notice that the user interface is mostly point to type although commands still refer to the place where the user made the last section. Once you get used to this, it is convenient because any command shown anywhere can be used to work on the file you are working with (it is very likely you selected something on it recently).

Olive owes much of its user interface to Acme [5], including sending characters

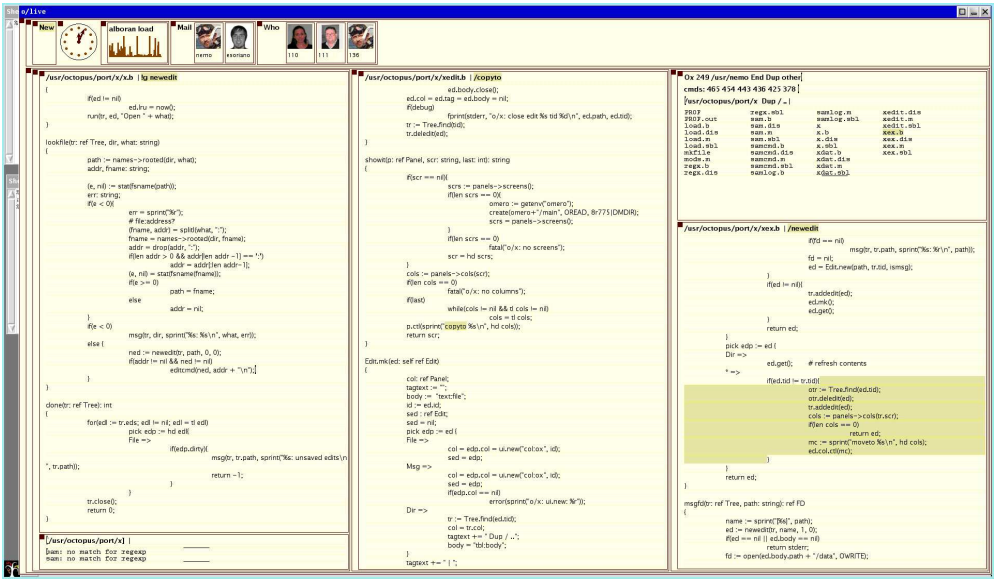

Figure 6: *Native tools using Omero at the PC.*

from the keyboard to the panel where the mouse is pointing to. But it is not Acme. It is more similar to Sam [3] in that it is customary to keep a commands panel to keep the text for several commands and use them at *any* file shown by the interface.

Another problem is that the user may execute `Close` on a panel containing most applications (e.g., the top-level one) and cause the user interfaces to be removed. This terminates most applications, unless they have interfaces shown at other screens as said later. If you are not lucky and kill Ox in this way, you may need to restart the PC to obtain a new file browsing and execution environment. This might be fixed in the future, but as of today, it is not.

5. Multiple screens

There is no need to keep all the programs (or terminals) using the same screen. Executing

```
Dup other
```

creates an additional screen, named `other` that can be used with Olive. When more than one screen exists, Olive asks the user which one should it attach to before doing anything else.

In any case, all terminals share the clipboard no matter if the screen they show is the same or not. Thus, cut and paste operations behave as expected. As an additional help, mouse and keyboard redirectors are used when several machines share a desktop so that a single mouse and keyboard suffices to use them (for Plan 9 they are available along with other Plan 9 software for use with the octopus in the main distribution site.)

The command language helps to handle applications when there are multiple screens. For example, the command `P` retrieves a list of the application interfaces matching a pattern (all of them if no pattern is given). It is similar to the command `X` used for files. Within `P` it is feasible to call an additional command to do something to the panels selected. For example,

```
P /main.*stats/D
```

would delete all panels with names matching the pattern given. This pattern would

select all panels in the statistics row (the top row) of the main screen. In a similar way,

```
P /main.*stats/ e /other/row:stats
```

replicates the interfaces shown in the main statistics row to the statistics row of the screen named `other`. This is most useful after creating a new screen, to copy the interfaces from desired applications to it, instead of executing multiple copies of the applications involved.

To terminate an application you may either kill it or close its user interface (using the command language or the `Close` menu option at its interface). Applications with replicated interfaces are not aware of the replication. When the last copy of the interface is gone the application receives an notification and usually exits.

6. Further information

All the details about how to use Omero and Olive are provided in the following manual pages:

olive(1) is an introduction to the window system and describes the mouse and keyboard commands available. If is a good reading after completing this tutorial.

ox(1) describes how the window system can be used to browse files and execute commands.

omero(4) is the reference to the file interface provided by the window system. It is useful to learn to use that interface and to discover control requests that may be issued also from the user interface.

For the most part, that is all needed to use the system. However, there are other tools documented in the Octopus user's manual. Most of them are Inferno modules and commands. In particular, section 4 of the manual describes file servers most of which are drivers available for the Octopus. In general, drivers and communication software is implemented on Inferno.

The Sam tutorial [6] is an excellent way to learn to use the command language. Only a few commands provided in some other way are missing from it, and only a few commands have been added as said in *ox*(1).

Finally, there are some applications and a few libraries for use on the native system. They are new programs that exploit features introduced by the Octopus into the native environment of the PC. They are documented in the appropriate section of the Plan 9 manual used on the PC environment.

References

1. F. J. Ballesteros, P. Heras, E. Soriano and S. Lalis, The Octopus: Towards building distributed smart spaces by centralizing everything., *UCAMI*, 2007.
2. S. Dorward, R. Pike, D. L. Presotto, D. M. Ritchie, H. Trickey and P. Winterbottom, The Inferno Operating System, *Bell Labs Technical Journal 2*, 1 (1997), .
3. R. Pike, The Text Editor Sam, *Software, Practice, & Experience 17*, 11 (Nov. 1987), 813–845.
4. R. Pike, D. Presotto, K. Thompson and H. Trickey, Plan 9 from Bell Labs, *EUUG Newsletter 10*, 3 (Autumn 1990), 2–11.
5. R. Pike, Acme: A User Interface for Programmers, *Proceedings for the Winter USENIX Conference*, 1994, 223–234. San Francisco, CA..
6. R. Pike, A tutorial for the Sam command language.