Considerations about a client-server approach for graphics

L.Chiappetti - IFCTR - Tue, 19 Jan 1993
(DAWG-REP.1.0/93)

1. Introduction

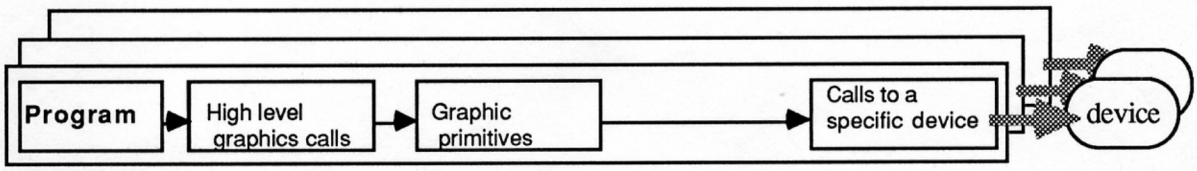
This report presents a very rough prototype of a set of client-server programs for graphics. The main advantage of the client-server approach is modularity : all programming efforts concerning a particular device are confined in the server handling that kind of device, and do not affect all other application developers (e.g. only one person needs to know about C language and X-11 programming to write the server, and all other people can go on writing their own simple graphics programs in Fortran).

The report first discusses some general concepts (some of which are implemented in prototype programs of general utility), then gives technical details on the specific implementation (at the level of prototype VOS library calls). A section follows presenting a demo server using PGPLOT. Since this is felt valuable as demonstration only, but not satisfactory as a production tool, a discussion of possible future development follows.

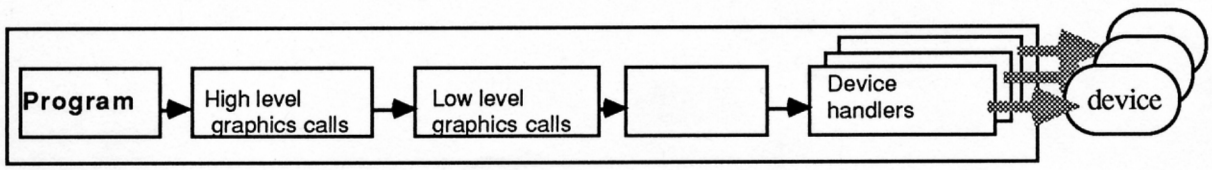
2. Client-server concept

The following schemes illustrates some common approaches in graphics programs and libraries, together with the relatively little used client-server approach.

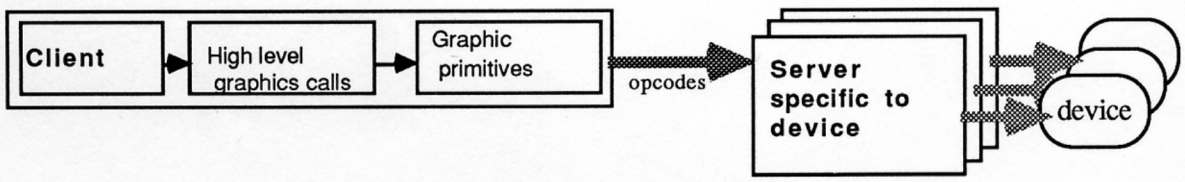
Multiple programs linked with a device specific library



Single program linked with all device handlers



Client-server approach



Document for which no computer readable source exists any longer

Partial scan of original hardcopy of 16 pag. (available on request) supplied

The traditional approaches are either to provide a set of libraries which is specific of every family of plotting devices, or to have a multiple-device library. In the first case one needs to link a separate copy of the program with each device library. In the second case (which is nowadays more popular, due to the relatively low impact) there is a module which distributes the calls to the specific *device handler* (sometimes incorrectly called *driver)* of the selected device, but all handlers (even if not used) are linked in the program.

The client-server approach allows more modularity in the development. Graphics programs (clients) need only to link with a common library, which sends primitive commands (in the form of some conventional opcodes and arguments, with the same form for all devices) over a communication channel to a server program. Each kind of device can have its own server program. Only the server needs to be linked with the device specific routines, or otherwise be able to generate device-specific code.
Also the know-how about a specific device is limited to the person developing and mantaining a specific server, freeing all other people from the need to worry about these items.
It is also possible at any time to add a new server without the need to relink any of the client programs.

An additional requirement is the possibility to keep more instances of the same server. This is not always exploited by other software packages which use a client-server approach (though maybe not fully consistently, for historical reasons). E.g. IRAF uses either imtool or SAOimage for image display, but keeps a single instance of them (one can run more, but they all will display the same stuff, since there is only one communication pipe).

2.1. Creating a server

A server must be running "in background" and keep alive any device (typically a window in X-11) so long it remains running. However *the server must be started by an user command.*

One can imagine to have *different types of servers* active at the same time (e.g. an `xwserver`, which manages an X-11 window; a `psserver`, which manages black&white Postscript files; a `cpserver`, which manages color Postscript; a `pgserver`, which runs via PGplot, etc.). In the prototype each server will be a separate program, named `xxserver`, where *xx* is a two-letter code.

One can also imagine to have *different instances of the same type of server* (i.e. different copies of the same program) running at the same time (e.g. each one to keep alive a different window). The prototype assumes each copy of a given type of server to be identified by a 1-digit number *n* between 1 and 9 (so there is a maximum of 9 servers per type, which is far more than needed).

Finally each server may require *extra arguments* to be started (e.g. a window server may require the position and size of the window, a Postscript server may require the page orientation, etc.). Although foreseen, passing this information to the server is not implemented.

The user command which activates a given server is called `createserver`. It will be called (runstring or interactively, usual XAS user interface) as `createserver` *`xx n args`* and will perform the following tasks :

it will check if the server is registered
it will check if the communication channel with the server is set up

The following status diagram indicates in **boldface** what is changed by the program.

be INTEGER values, while arguments can be INTEGER or REAL. They all will be sent via Fortran binary unformatted i/o.
The main design task is therefore the selection of the primitives.

5.6. Let the server do the scaling ?

One idea, indicated by the slogan "let the server do the scaling" (and let's hope that British Telecom won't sue us for our cast on their Yellow Pages slogan of a few years ago!) is that graphics clients can send coordinates indifferently in world coordinates, normalized device coordinates (NDC, 0.0-1.0) or true device coordinates (and perhaps also cm or inches). The client will just need to send a "set coordinate" opcode in advance, and the interpretation of the coordinates will remain what set until another such opcode is sent.
Since all coordinate conversion is done in the server, this will also allow to set up things like logarithmic scales (or even other kind of coordinates, like polar, or spherical projections) in a way which is very easy for the user.

Each server will anyhow set up a default viewport (either the entire physical page or window, or a predefined fraction of it). The server will usually clip graphics which falls outside the viewport, but allow text to go anywhere. Specialized opcodes will allow to change viewport and window.

5.7. Possible list of primitives

The following represents a possible list of primitives. They are divided in three cathegories : graphics actions, set mode and query mode.

Opcode	Primitive	Arguments	Note
-2	server directive	directive-code + variable	send a server-specific directive
-1	termination	none	terminate a server
0	disconnect	none	signal client is going to disconnect
1	clear	none	clear screen or advance page
2	move	x y	move pen to x y
3	draw	x y	draw to x y (necessary ?)
4	polyline	n x(n) y(n)	plot a polyline of n points
5	polymarker	n x(n) y(n)	plot markers or points (*)
6	polyfill	n x(n) y(n)	fill a polygon (*)
7	text	length text(length)	plot text (*)
8	clear area	none	clear viewport
9	write image	length data(length)	write image data in viewport
10	write lut	start n r,g,b(n)	set up colour lookup table
11	read cursor	none ?	will return cursor position plus other info on the output channel
12	read image	??	will read image data back on the output channel
13	read lut	??	will read colour lookup table back on output channel
101	set viewport	x1 x2 y1 y2 (NDC or device)	set position and size of viewport
102	set window	x1 x2 y1 y2 (world)	associate window to viewport

103	set coordinate	NDC | device | world or also cm | inch	set units for all following coordinates
104	set scales	lin log polar etc. TBV	set coordinate conversion
105	set pen colour	colour	single or separate for lines, text, fill ?
106	set linewidth	width TBV	set line width (*)
107	set linestyle	e.g. solid, dashed etc. ?	set line style (*)
108	set background colour	colour	TBV if needed, perhaps images only ?
109	set marker	marker number	set marker for all following polymarker calls
110	set text font	font code (mapped from a `local` file ?)	select font for text (*)
111	set text size	size TBV	set size of text (*)
112	set text orientation	angle in degrees	set text orientation (*)
201 to 212	query calls	TBV	they correspond to the set opcodes (101-112) and return on the output channel the requested info with the same format as the set call

(*) All items marked with an asterisk are intended to be implemented (at least initially) "in hardware", that is only if the hardware, firmware or system software support it. For instance text will not be drawn using software in our own graphics library, but using system fonts (X-11 or Postscript, or plain terminal hardware characters, in which case some characteristics are not supported); polygon filling (of dubious usage) or variable line width will be done only if already implemented (e.g. by X-11, Postscript or HP-GL), etc. etc.