Solaris VISUAL 2

The Solaris Computing Environment

The Solaris VISUAL graphics environment is the graphics rendering component of the Solaris operating system. The Solaris operating environment includes:

- SunOS™, a 32-bit UNIX operating system based on SVR4.
- ONC+, the standard for heterogeneous networking technology.
- OpenWindows, SunSoft's X Windows-based window system.
- DeskSet™, an integrated desktop including 15 productivity tools.

SunOS is the foundation of Solaris. Solaris is based on UNIX System V, Release 4, the industry's leading standard for an open, scalable operating system environment. SunOS features symmetric multiprocessing with a multi-threaded kernel, real-time extensions, Application Binary Interface (ABI) compatibility for SPARC and x86 architectures, advanced internationalization capabilities, security and simplified system administration.

Solaris ONC+, the latest implementation of the industry standard Open Network Computing environment, includes the NFS (Network File System) distributed file system, the industry standard for transparent remote and local file sharing between heterogeneous systems.

DeskSet includes personal and workgroup productivity tools such as a multimedia mail tool, calendar manager, audio tool, text editor, snapshot tool, calculator, clock, file manager, print tool, tape tool, performance meter, binder (for customizing desktop applications), icon editor, shell tool and command tool.

OpenWindows is an MIT X Windows-compliant distributed window system based upon MIT X11 Release 5. Components of OpenWindows include an X Window Manager, several graphical user interface toolkits, the OpenWindows Developer's Guide user interface prototyping tool, and the ToolTalk inter-application messaging service for desktop integration.

OpenWindows provides the base platform in which the Solaris VISUAL graphics environment interfaces run. OpenWindows is a distributed, client-server window system permitting the remote display of applications within it. The flexibility of the client-server model in supporting graphics enables applications to be run and/or displayed across a network based upon the users needs or desires. As true enterprise computing becomes a reality, this flexibility will increasingly become important to a wide variety of users.

Figure 2.1 illustrates the Solaris and its components.

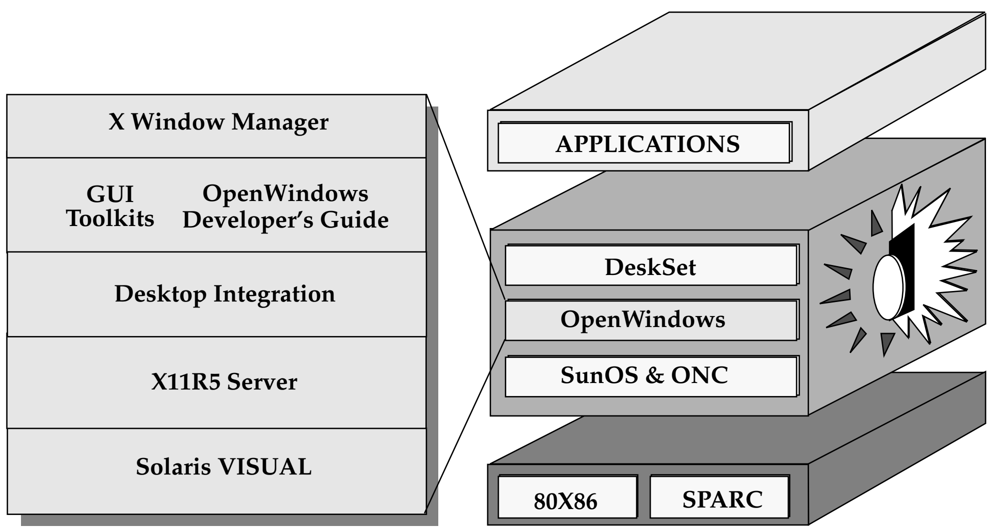

Figure 2.1 Solaris and its Components

The Solaris VISUAL Environment

The Solaris VISUAL environment consists of a set of foundation libraries providing support for bundled X graphics libraries and third-party graphics APIs. The foundation library interfaces address the major graphics rendering requirements of geometry, imaging, PostScript and pixel rendering. The foundation libraries provide for the breadth of functionality required by industry standard and third-party APIs.

The X graphics libraries provide APIs for local or network-distributed execution and display of graphics applications.

All of the foundation libraries are architected to support Graphics Porting Interfaces (GPIs) for the simplified integration of third-party graphics devices. Applications using any of the Solaris VISUAL components or third-party APIs will work with any Solaris VISUAL display device transparently, thus reducing porting and maintenance efforts on the part of developers, and freeing users to select the best hardware for their particular requirements. Figure 2.2 illustrates the Solaris VISUAL environment.

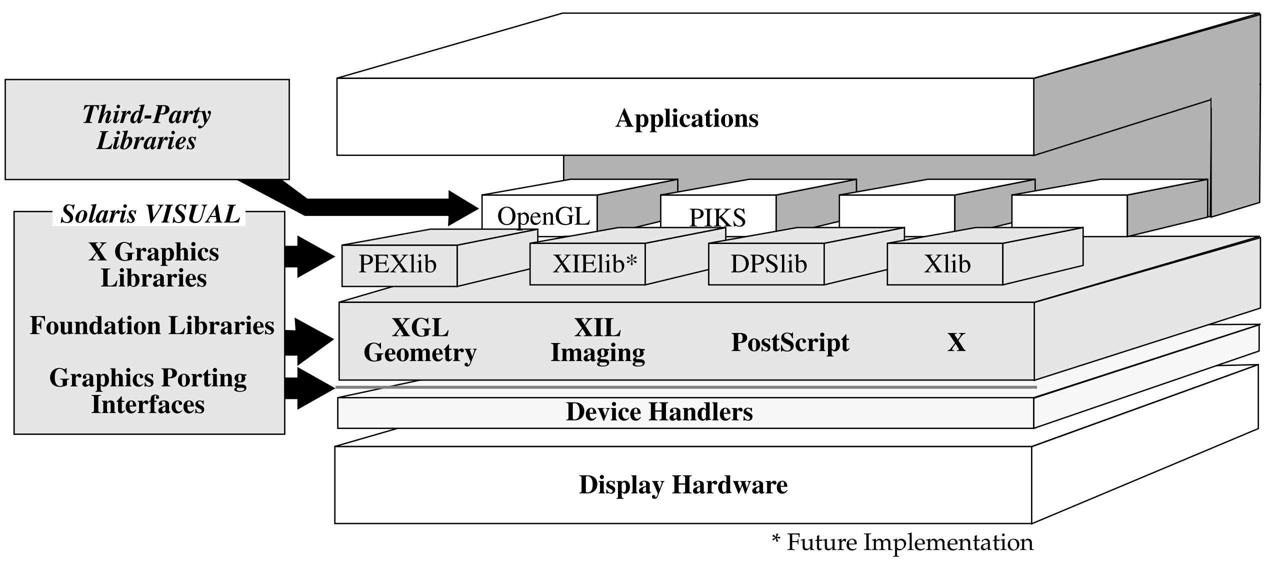

Figure 2.2 Solaris VISUAL

Solaris VISUAL Components

Each of the Solaris VISUAL foundation libraries has a corresponding X graphics library and network wire protocol. The X graphics libraries are industry standard bindings to protocols defined by industry or de facto standard organizations.

The table in figure 2.3 below illustrates the relationship between individual Solaris VISUAL foundation libraries, their industry standard X graphics library, and their associated wire protocol.

Graphics Rendering Approach	Solaris VISUAL		
	Foundation Library	Industry Standard (X Graphics APIs)	Wire Protocol
2-D/3-D Geometry	XGL	PEXlib	PEX
Imaging	XIL	XIElib	XIE
Stencil/Paint	PostScript	DPSlib	DPS
Pixel-based Rendering	X	Xlib	X11

Figure 2.3 Solaris VISUAL Components

Each of the Solaris VISUAL foundation libraries was designed to support a variety of hardware-based functionality. Mapping of the foundation libraries to underlying hardware is achieved through flexible device porting layers known as the Graphics Porting Interfaces. Figure 2.4 illustrates how the Solaris VISUAL components are layered.

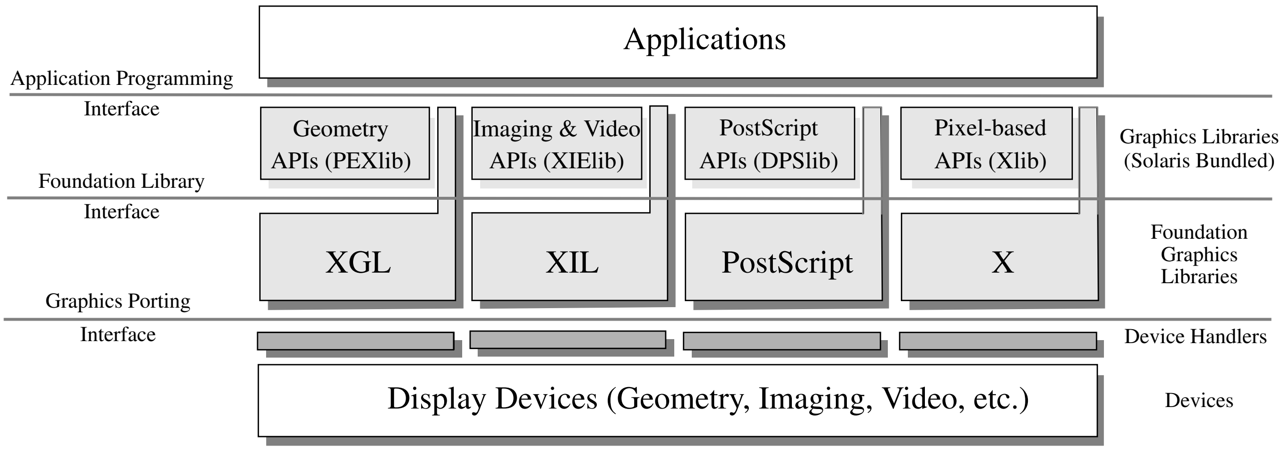

Figure 2.4 Solaris VISUAL Components: Layering

Solaris VISUAL Device Independence

Device independence of graphics applications and APIs in the Solaris VISUAL environment is made possible by layering described previously. Each Solaris VISUAL foundation library provides a graphics porting interface that the hardware developer may port to the device.

Only devices which provide architectural or performance benefits to a given foundation library (rendering method) need be ported to that library. For example, it is probably unnecessary to port a device specialized for image compression/decompression capabilities to the geometry foundation library, XGL. And some sophisticated geometry accelerators are not the best match for porting to the imaging interface, XIL.

Only one graphics porting interface is absolutely required for porting: the X Windows graphics porting interface. All other foundation libraries support a default device handler which maps to the X Window system rendering layer. Hence if a hardware developer of a simple device wants to make a product available under Solaris with the minimal amount of effort, he/she needs only port one interface. If however, the device has special characteristics or features, it probably makes sense to port one or more of the other foundation library graphics porting interfaces.

Figure 2.5 illustrates the graphics porting layers for each of the foundation libraries. The sections which follow describe the graphics porting interface architecture in more detail.

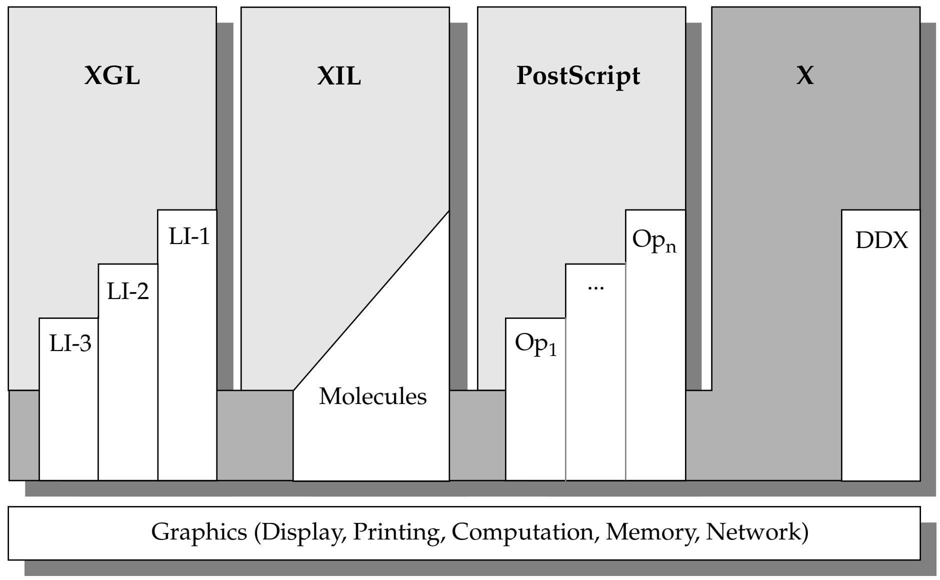

Figure 2.5 Solaris VISUAL Hardware Vendor View: Device Porting Layers

The XGL Foundation Library

The rendering pipelines include one or more device pipelines and a software pipeline. The pipelines are divided into three layers, each responsible for interfacing at specific pipeline functions. The top layer, Loadable Interface 1 (LI-1) specifies a device-independent interface that lies directly below the XGL API. Functions in this layer implement the geometry pipeline for each drawing primitive; they take the points defining the primitive and transform, light (in the 3-D case), and clip the geometry in preparation for the rendering operations in the next layer. The second layer, Loadable Interface 2 (LI-2) is responsible for scan converting more complex primitives like polygons and polylines. The lowest layer, Loadable Interface 3 (LI-3) is responsible for rendering pixels, individually or in spans, on the device.

The default pipeline is the software pipeline which XGL provides for devices that have only minimal (or no) acceleration capabilities. The software pipeline consists of functions that implement the geometry pipeline (LI-1), and scan conversion (LI-2) functions in software. A device pipeline can choose to override the default software implementation for any primitive, choosing instead to accelerate the primitive's rendering through the accelerator's own facilities.

Figure 2.6 illustrates the XGL pipeline architecture.

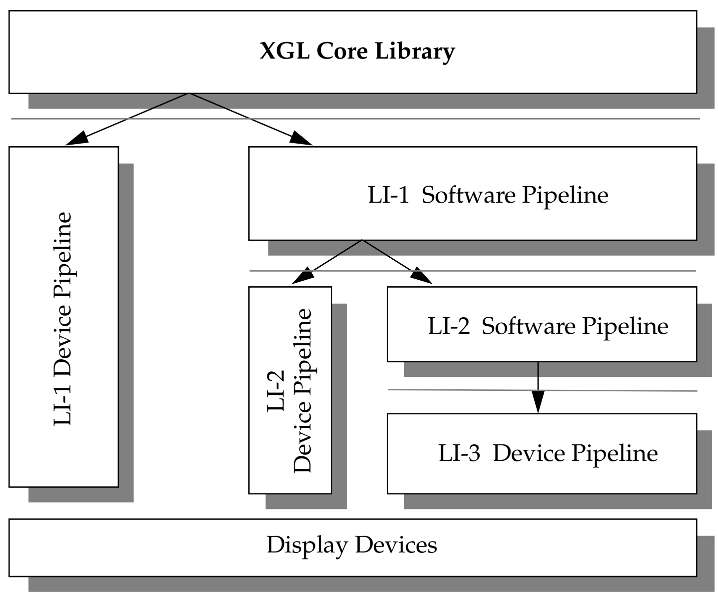

Figure 2.6 Solaris VISUAL: XGL Device Porting Layers

The XIL Foundation Library

Imaging and video applications may call XIL directly or utilize de facto or industry standard APIs which may be implemented above XIL. Developers of display devices, such as framebuffers, imaging accelerators and video products would write a loadable device handler which maps the functionality of their device to the XIL hardware specification. Figure 2.7 illustrates the XIL architecture from a software and hardware developer's perspective.

The graphics porting interface employed with the XIL foundation library provides a flexible mapping of functionality between XIL and a wide variety of imaging devices through access to XIL's native atomic operators. Every XIL operator can be considered an atom. Groups of XIL operators, concatenated together in a sequential manner are known as molecules.

Developers implement XIL atoms and molecules based upon the capabilities of the target devices. For example, a developer whose device performs geometry processing, such as rotation or bilinear scaling of images might choose to implement only geometric functions. If a device supports a variety of geometric and image processing functions, the developer may elect to implement atoms or molecules from both functional areas.

Molecules are particularly important to device developers because they allow the device to notify XIL at runtime that the device wishes to provide hardware support for groups of chained operations. For example, a specialized imaging processor might provide image scaling and casting from 16 to 8 bit data. If the device only provided image scaling, it would have to move the data back to CPU memory for XIL software casting from 16 to 8 bit. The additional copy operation is avoided if the device can scale and cast on-board.

At runtime, the device handler loads into memory and notifies XIL of the atoms and molecules it wishes to replace with its own mapping to hardware. XIL uses a direct acyclic graph to hold a list of these dependencies during XIL application execution. When XIL encounters a known sequence of operations (defined by a molecule), it transfers the operation to the device for execution.

The XIL foundation library graphics porting interface supports three basic device handler types: Compute handlers, Storage handlers, and I/O handlers. Compute handlers provides accelerated functions for image processing and compression/decompression. Storage handlers manage data in their own device memory, usually during manipulation of the image. I/O handlers are required for all frame buffers and image capture devices which perform I/O.

Developers can implement one or more device handlers dependent on the capabilities of the device.

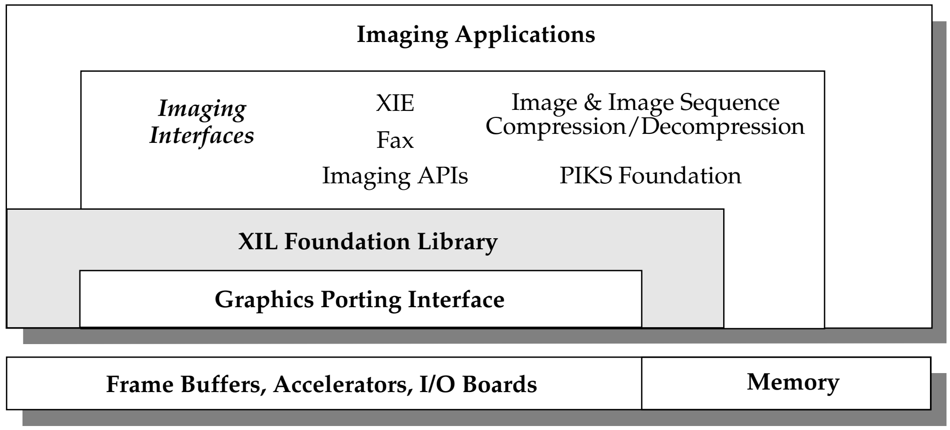

Figure 2.7 Solaris VISUAL: XIL Foundation Graphics Library Architecture

The PostScript Foundation Library

The PostScript foundation library is SunSoft's optimized implementation of Adobe System's Display PostScript architecture. Hardware device developers have several choices in how they support PostScript rendering.

A device developer can port just the DDX interface (described in the section: X Foundation Library), to their device and all Solaris VISUAL graphics libraries will work, including Display PostScript.

If the display device is directly accessed as a memory mapped framebuffer, increased PostScript rendering performance may be achieved by enabling Adobe's direct marking capability.

To utilize the direct marking capability on a given device, the device developer must develop a device handler. At initialization, the device handler provided by the developer provides a pointer to a structure which contains pointers to device dependent subroutines (provided by the vendor). SunSoft defines the format of the structure which the developer must follow. The PostScript foundation can then call the routines provided by the handler to perform PostScript rendering.

In the future, SunSoft and Adobe will enhance the PostScript GPI to support even greater device customization.

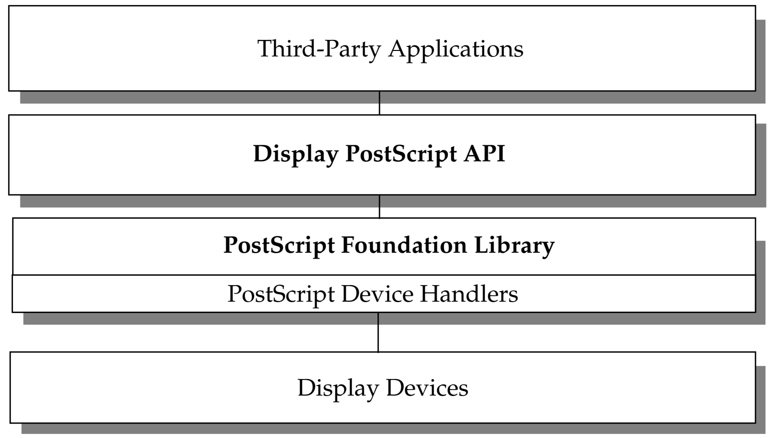

Figure 2.8 Solaris Visual: PostScript Foundation Library

The X Foundation Library

The X Window System server consists of several parts including the device-independent (DIX) layer, the device-dependent (DDX) layer, the operating system (OS) layer and the font management library.

The DIX layer contains the portions of the server that are device independent and portable from one implementation to another. The DDX layer contains code which is dependent upon the capabilities of the display and input devices attached to the system. The OS layer includes server functions that interact with the native operating system in which the server runs. The font management library contains collections of routines that manage the location and loading of text fonts.

Hardware developers wishing to port the OpenWindows X server to their device concern themselves principally with the DDX layer. The DDX layer is responsible for creating and manipulating pixmaps, clipping regions, colormaps, fonts, screens and graphics contexts. The DDX layer also collects events from input devices such as the keyboard and mouse, and sends them to the DIX layer.

Upon X server start-up, the DIX layer determines the display device type, and loads the corresponding device handler. The DDX layer must then provide a screen initialization routine to the DIX layer to use. This routine returns a screen object with a vector of handler-provided function pointers.

Handlers can substitute their own routines or reference device-independent routines, dependent upon the device's functionality. The DIX layer additionally provides a set of utility routines to the DDX layer device handler. These routines include facilities for colormap management, multiple plane group support, and initialization of memory-mapped frame buffers.

Hardware developers familiar with the X Windows DDX layer will find the Solaris VISUAL X foundation library graphics programming interface identical in most respects. SunSoft has documented the interface in substantial detail to ease porting, and has defined more stringent requirements for devices to insure their continued portability.

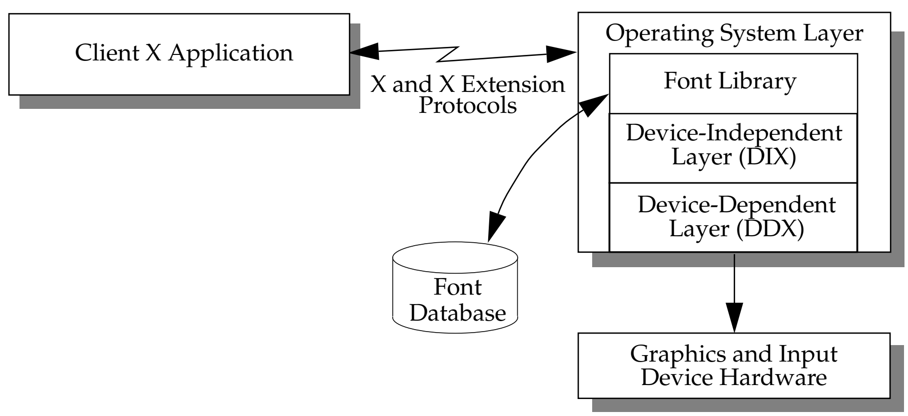

Figure 2.9 Distributed Imaging Application Under X Windows

Developing and Running Applications under Solaris VISUAL

To develop and run applications on many operating systems, the developer must manage a complex process insuring that the graphics interface is available on the users system, sometimes requiring special licensing and shipping arrangements on the part of the product developer.

Solaris VISUAL gives the developers the option of using either the foundation libraries or the X graphics libraries or co-bundling their selected 3rd-party APIs. Developers have much more freedom in matching their application's requirements because Solaris VISUAL interfaces are designed to support the broadest range of industry and de facto standard APIs.

Solaris VISUAL Runtime Support

The Solaris VISUAL runtime environment ships on every Solaris-based system. Users can select from many thousands of applications and be confident that they will run on their graphics display.

The Solaris VISUAL runtime support includes all of the foundation libraries — XGL, XIL, PostScript, and X — and the X graphics libraries — PEXlib, XIElib, DPSlib, and Xlib.

OpenWindows Server Capabilities

Many of the graphics libraries utilize SunSoft's Direct Graphics Access™ (DGA) technology to directly render via hardware accelerators whenever an application is running locally. The X interprocess communication mechanism is bypassed to permit high-performance rendering and a more direct mapping of functionality between the graphics library and the accelerator hardware.

The OpenWindows server also provides other graphics-related features, including:

- Multiple visual type support, including 8-bit PseudoColor, StaticColor, GrayScale, StaticGray, and 24-bit DirectColor and TrueColor
- Access to hardware double buffering
- Support for additional peripherals, such as buttons, dial boxes, joysticks, etc.

Running Applications Across the Network

The client-server graphics model used by OpenWindows and Solaris VISUAL works across the network as well on the same machine. In a pure client-server implementation, the application client communicates with the display server of the X Window System through one or more of the X Protocols: PEX, XIE, DPS and X11.

Third-Party APIs may provide client-server capability whether they are layered above the foundation libraries or the X graphics libraries. In each case the application can use the capabilities of the X Window System to distribute the application. Remote rendering is handled transparently by use of the X protocols. Figure 2.10 illustrates a Solaris VISUAL client application and the X protocols..

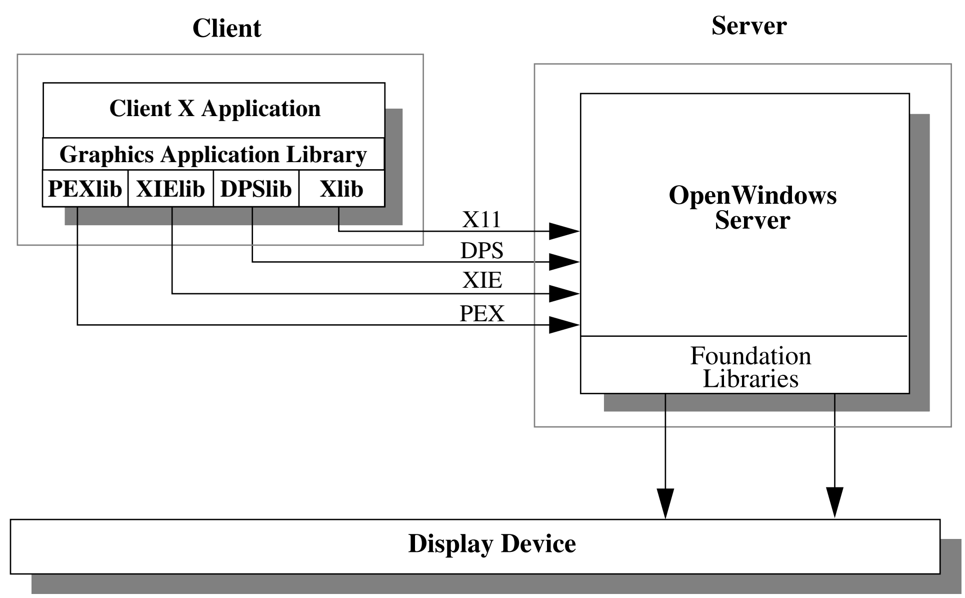

Figure 2.10 Solaris VISUAL Interfaces and their Network Protocols

Direct Graphics Access™ (DGA)

SunSoft designed DGA into the OpenWindows server to enable locally rendered graphics to fully utilize the functionality and performance of graphics accelerated framebuffers and processors. When the client application and the server are on the same machine, foundation libraries and interfaces layered above them use DGA to communicate and synchronize with the X server while accessing the graphics hardware directly, thus maximizing throughput between the graphics client application and the hardware. Currently, any API or application using XGL, XIL or Direct Xlib implicitly takes advantage of DGA.

DGA efficiently overcomes performance constraints associated with the Inter-process Communication (IPC) mechanism employed with the X protocol. The IPC bottleneck has inherent overhead associated with extra copy operations between the client application, the network connection and the server.

The use of DGA is transparent to the application developer. Whenever an application runs on the same machine as the server, XGL, XIL and Direct Xlib detect it and invoke DGA to coordinate display sharing between the client and the window system.

Figure 2.11 illustrates Solaris VISUAL's relationship to DGA and the server. For example, when an XGL application creates a Raster Device object, XGL automatically determines whether DGA is available and whether XGL supports the display device. If DGA is available and the device is supported, XGL synchronizes with the server to share direct access to the display. In this case, XGL renders graphics geometry to the display device, while the server continues to perform other window operations. Figure 2.11 illustrates the DGA mechanism and its relationship to the Solaris VISUAL components.

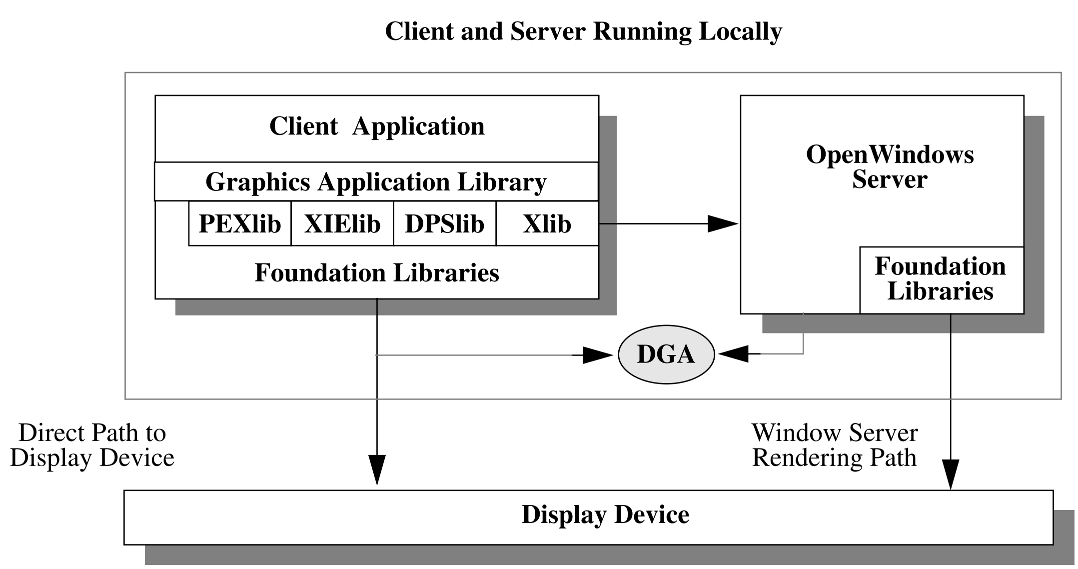

Figure 2.11 DGA and Solaris Visual Interfaces in the OpenWindows Environment