The Hewlett-Packard Approach to Dynamic Loading within the X Server

Ronald C. MacDonald

The HP X server implementation supports dynamic loading of HP and third-party hardware and software products using a plug-and-play approach that maintains X server robustness.

The goal was to simplify creation and support of new products while eliminating required end-user configuration responsibilities for the X server. This was accomplished by addressing design limitations in the X Consortium's sample server. The HP solution introduces two new paradigms called *broker* and *coinitialization*.

The broker paradigm is the backbone of the HP dynamic loading solution. This mechanism:

- Provides standard interfaces to support third-party products
- Manages graphics hardware, associated software DDX drivers, and X extensions
- Supports order independence of product installation
- Permits restriction of end-user configuration to noncritical feature information
- Reduces system resource requirements by deferring loading X extensions until they are used
- Maintains a log of selected components and configuration information for reproducibility and maintenance
- Handles product configuration requirements and clarifies product revision issues
- Introduces a new X server startup control point, creating a very flexible and extensible server initialization process for product developers.

Ronald C. MacDonald
With HP since 1988, Ronald MacDonald is a software engineer at the HP Workstation Systems Division. He was a member of the X server team responsible for dynamic loading development and third-party partnerships. He has a BS degree in forestry (1975) from the University of Michigan and an MS degree in computer science (1986) from the University of Arizona.

The coinitialization paradigm defines a straightforward process for joint initialization of graphics drivers and X extensions requiring graphics driver support. Solving a classic chicken and egg dilemma in which the graphics driver and X extension both need the other in place to complete their initialization.

Background

The term "X server" refers to the X Window System server developed at the Massachuset Institute of Technology and then managed by the X Consortium. The X server is the functional basis for windows on workstations running the UNIX® operating system. What appears on the computer screen is the output generated by the graphics card in the workstation managed by the X server.

Over time the quantity of supported graphics hardware devices and X extensions has grown to the point where providing a single binary to support everything is not practical. In addition, some X extensions do not work with all graphics cards, and third-party graphics cards and extensions need support. Since the end user can change the workstation configuration after initial shipping, a more flexible approach is mandated that can determine a workstation's configuration and select a subset of the binaries present to support that configuration.

This article discusses how the HP X server implementation addresses configuration issues encountered during server initialization. This includes honoring optional end-user configuration requests, selection of specific graphics hardware and binaries from an available pool, and accommodation of a continuously evolving set of X server extension binaries. X server extensions are enhancements or modifications to the core functional X server.

The HP X server implementation accomplishes this integration by separating the various components into dynamically loaded modules so that all components can be considered with only a minimal set of binaries loaded to form the run-time binary for any given X server instantiation. The set of routines that performs this selection and integration is referred to as the *collector*.

Broker Paradigm

The broker paradigm is a very simple concept with surprising capabilities. Brokers represent each dynamically loaded X server component. Brokers are shared libraries created by the product vendors. The brokers are called upon by the collector, which interrogates them individually while sharing information among all of them. The collector determines which brokers are eventually selected for loading based on information provided by the brokers as to what they are and what needs to be loaded. Other than product installation, no further end-user action is required because the X server loads the correct brokers during initialization.

During X server startup, initialization steps occur to fill the X server data structures. Network sockets are established, input and output devices are set up, and X extensions are initialized. All this is done to prepare for entering the dispatch loop to enable client requests to be processed by the X server.

The broker paradigm is executed as part of this X server initialization phase. First, end-user requests, such as which graphics card to use, are checked for. The requested hardware is tested and identified. If problems are encountered, defaults are substituted in an attempt to start up the X server. Since the X server is the primary human-machine interface, robustness is required.

Next, all the extension brokers present are queried via standard interface definitions to learn their properties and graphics requirements. Each broker contains the traditional product definition and configuration information. This information, along with target graphics hardware descriptions, is then presented to all graphics brokers. Each graphics broker reviews the information provided and responds with a bid for the hardware and a vote on each extension. Graphics brokers can represent one or more graphics drivers capable of supporting specific graphics hardware with varying X extension support capabilities.

The collector evaluates the hardware bids, selecting the highest bidder for each target graphics card. Graphics broker bids are based on the ability to support specific hardware, performance optimizations, and product revisions. After the graphics hardware drivers are selected, the extension votes from the winning graphics brokers are examined to determine which extensions can be supported.

After the target dynamic components have been identified, the selected brokers are again queried to determine which shared libraries should be loaded to define the selected products. At this point it is known which products

will be loaded. This information is shared with the selected brokers, providing a new control point in the X server startup. This control point permits the selected brokers to review the given X server's instantiation definition and make final configuration decisions such as asking the X server to change shared memory size or identifying additional shared libraries for loading. With this last information available, the collector proceeds to load the identified shared libraries in preparation for the normal graphics and extension initialization steps. Finally, a log file for recording the configuration and loaded shared library information is maintained.

Since the collector has all the configuration information in hand, it is possible to delay loading extension libraries until the first protocol request for that extension is encountered. This can significantly reduce the system resources required for a given X server instantiation and reduce X server startup time. Some X server extensions are quite resource intensive.

Coinitialization Paradigm

In the X Consortium's sample X server, the graphics drivers are initialized before the extensions. No initialization interplay between graphics drivers and X extensions is addressed. Yet, with each X extension requiring graphics driver support, additional special control and information interfaces are typically introduced with joint initialization dependencies. Coupling this joint dependency with order independent product installation and dynamic selection of components provided by the broker paradigm, missing or new components become a likelihood. Fortunately, the HP coinitialization paradigm provides a simple solution.

During screen initialization, graphics drivers check which dynamically loaded extensions are present. This involves searching for extensions that require specific graphics driver support to function correctly. When a particular supported extension is located, assuming the given graphics driver supports that extension, the required special initialization steps are taken as far as possible. Frequently, information must be obtained from the extension before initialization can be completed. (Remember the extensions have not yet been initialized.) This might involve initializing function pointers or allocating data structures. Additionally, a callback is registered with the X server for later use by the extension during its initialization.

Finally, during extension initialization, the graphics driver callback is retrieved. The extension can use the callback to request further actions from the graphics driver. Since the nature of the callback and its argument list is an extension-specific convention, the mechanism is very flexible.

Conclusion

These flexible paradigms not only simplify the HP product development and release considerations, they also make it fairly easy to partner with third-party vendors to place their products on HP-UX platforms. The standard interfaces make independent development straightforward and dynamic loading permits order independent product releases and installation. With all the complex vendor-specified configuration information located in the brokers and managed by the collector, the end user truly has a plug-and-play environment.

- Return to Article 7
- Return to Subarticle 7a
- Go to Next Article
- Go to Journal Home Page