Directions in Software Development in the PX Group at SSRL

Timothy M. McPhillips

Stanford Synchrotron Radiation Laboratory

Guidelines for Software Development in PX Group

Cross-Platform

- Multiple OS's needed at PX beamlines, future needs unkown.
- Use XOS (Cross-Operating System) Library for low-level software and Tcl/Tk for GUI components when feasible.

Distributed

- Applications must integrate services provided by different platforms.
- Use network interfaces between application components.
- Separate user interfaces from other components.

Reliable

- Program application components at highest level possible. No device drivers.
- Make it impossible for our software to hang or crash a computer.
- Handle all possible errors and use timeouts, retries, etc., as appropriate.

High-Performance

- Handle high-speed hardware synchronization within motion controllers.
- Distributed architecture lets us choose appropriate language for component.

Integrated with Existing Technology

- Use existing hardware/software when possible, new solutions when needed, integrating both via distributed/cross-platform development.

Documented

- Make installation distributions, source code and documentation available to other groups.
- Document well enough that other groups can use and extend our solutions on their own.

Need for Cross-Platform Development

Commercial Detector Hardware/Software Supported

- MAR imaging plate scanners (4 systems) require UNIX.
- ADSC CCD detectors (2 systems) require UNIX and Windows 95.
- What OS will our next detector purchase require?

Crystallographic Software Supported

- Two versions of UNIX required to keep users happy.
- What will users need next? Linux? Windows NT?

Beamline/hutch Hardware Requires Different Platforms

- ICS requires VMS.
- Motion controllers from Galil run under Windows NT.
- What will new SSRL control systems require?

Future of Operating Systems Unknown

- Futures of DEC, UNIX vs NT, etc. are open questions.

➔ **Only platform-independent programming can provide stability in this chaotic environment.**

Cross-Operating System Library (XOS)

Implementation Details

- **Compile-time approach.**
  - Header file xos.h loads appropriate, system-dependent include files.
  - Objects implemented as typedef structs, hiding architectural differences.
- **Restricted to portable, multithreaded, distributed programs.**
  - No non-network interprocess communication.
  - No graphics capability. Just system programming.
  - No unusual privileges required to call any function.

Advantages

- **Portability**
  - Compile code on Digital Unix, IRIX, OpenVMS, Windows NT/95.
- **Reliability**
  - Simpler programs leads to more reliable code.
  - Less need to study different platforms.
  - Normal user privileges means programs cannot crash computer.
- **Performance**
  - Native system calls on each platform for maximum performance
  - No runtime overhead for platform independence.

XOS Library Features

Threads

Mutexes

Counting Semaphores

- implemented using thread condition variables under OpenVMS
- timeouts supported

Thread Messages

- compatible with Windows messages
- built-in support for passing semaphores

Message Queues

- for sending text strings between threads

Memory-Mapped Files

- for single-process use only
- convenient and safe alternative to saving/loading binary files

Hash Tables

- looks up integer based on string
- can store function addresses for fast text message handling

TCP Sockets

- only provided method of interprocess communication

XOS Performance by Platform

	Mutex Lock and Unlock	Sema-phore Post and Wait	Message Queue Write and Read	Thread Message Swap	Socket Message Swap	Socket Message Swap (Burst)	Mailbox Message Swap	Mailbox Message Swap (Burst)
Bl710 Alpha 21064 233 MHz OpenVMS	2.6	5.7	20.0	74.4	324 (920)	258 (2425)	116 (300)	118 (315)
BIOPS Alpha 21064 233 MHz DEC Unix	2.0	4.8	15.6	125	168 (475)	63.6 (215)		
BL712 Alpha 21164 500 MHz DEC Unix	0.24	0.57	2.2	18.7	36.4 (105)	12.3 (45)		
BL713 MIPS R10K x 2 195 MHz IRIX	0.83	2.7	8.4	20	86 (255)	53 (1480)		
BIOTMPC Pentium II x 2 333 MHz Windows NT	8.5	7.9	34.4	70.4	261	1070		

Notes:

1. Values shown are average CPU execution times in microseconds. Numbers in parantheses are real times.
2. Thread message exchange includes two thread switches, socket and mailbox message exchanges include two process context switches.

DMC-1000 Controller from Galil Motion Control

1-8 Axes DC or Stepper

- DC motors need for Huber Kappa goniometer
- Encoder feedback supported for both types

16 Digital I/O Lines

- Fast shutter control.
- Additional limit switches.
- External keys, switches.
- Inputs can trigger interrupts.

7 Analog Input Lines

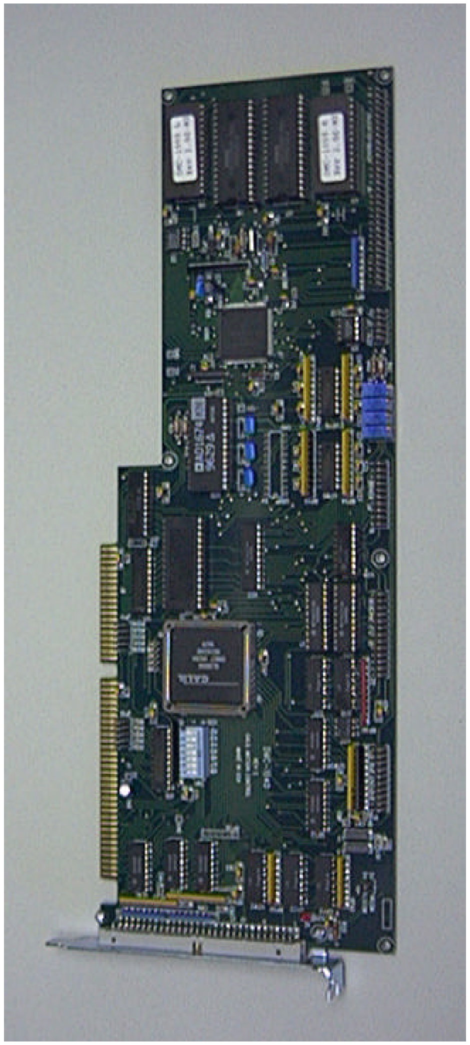
- Joystick control.
- Analog servo feedback.

Highly Programmable

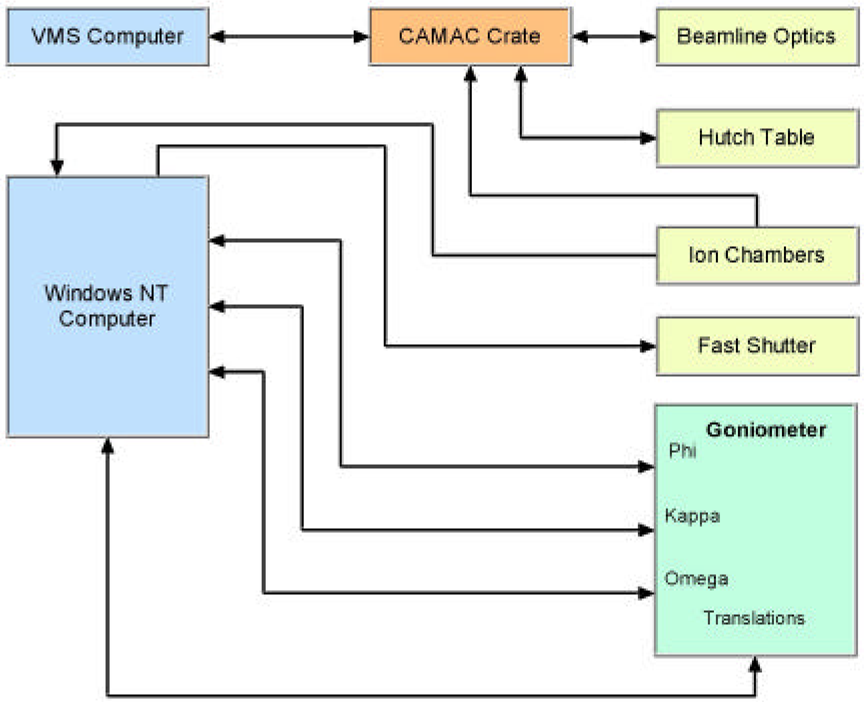
- S-curve profiling for smooth acceleration
- Synchronization of shutter and omega axis on-board

No Device Driver to Write

- ISA card for PC
- Device drivers available for Windows NT



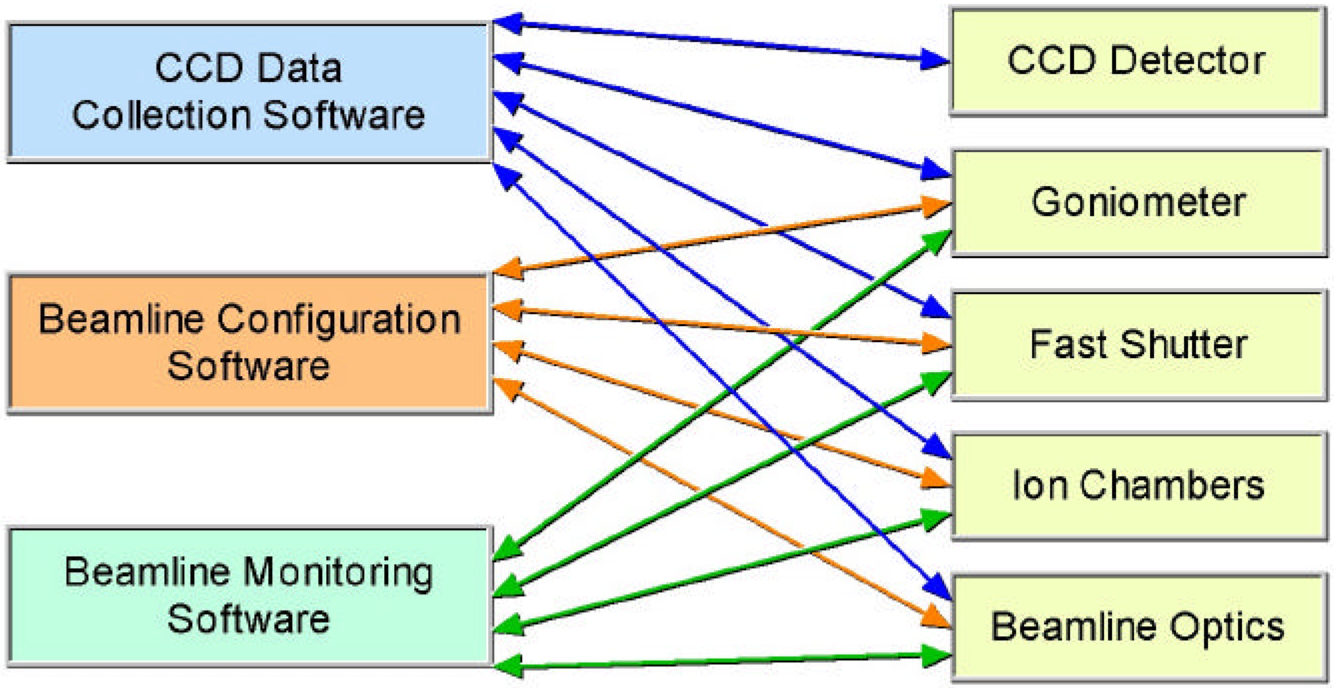
Problem 1: Multiple Hardware Hosts



➔ **Need Centralized Control of Beamline Components**

- Oversee operation of an arbitrary number of hardware hosts on multiple computing platforms.
- Maintain a single database of component positions.
- Coordinate motions and prevent collisions.

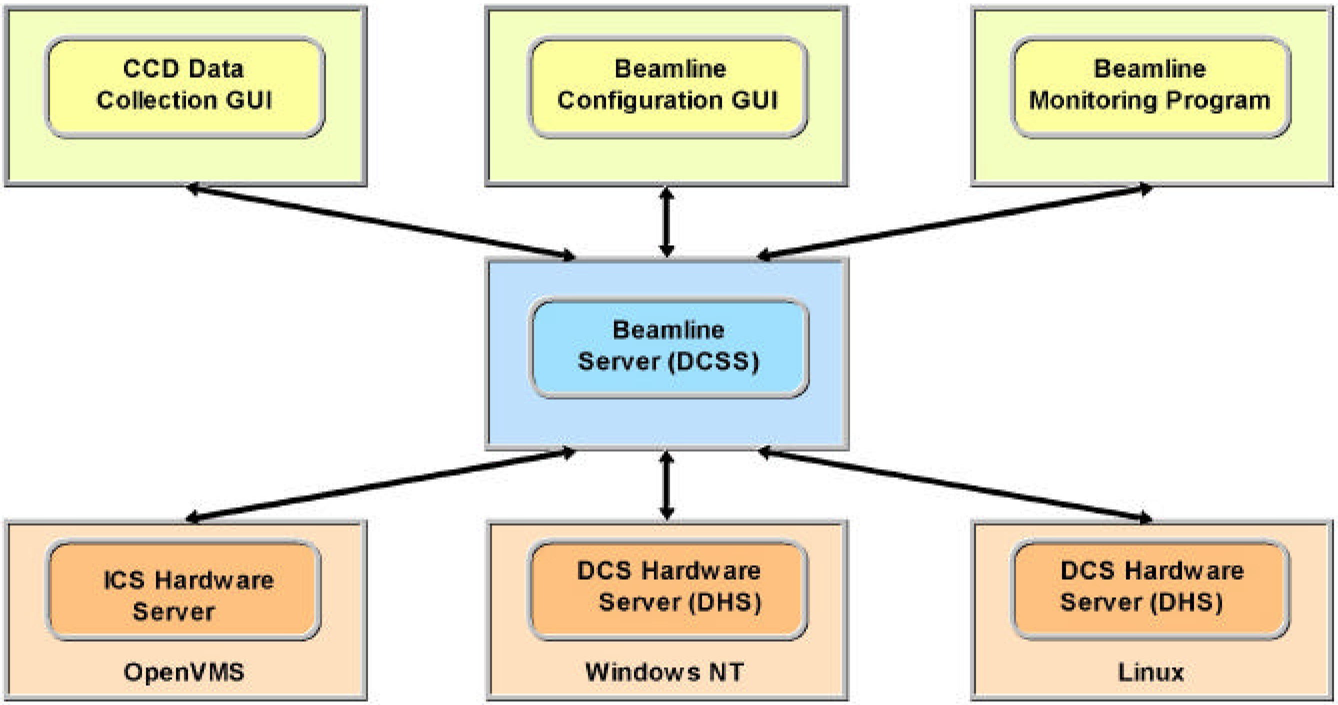
Problem 2: Multiple, Simultaneous User Interfaces



➔ **Need Centralized Authorization of User Interfaces**

- Prevent conflicts between user interfaces.
- Oversee transfer of control between processes.
- Allow interfaces to run anywhere on the network.
- Protect beamline from unauthorized access.
- Need *collaborative*, not *competitive* software.

Solution: Distributed Control System (DCS)



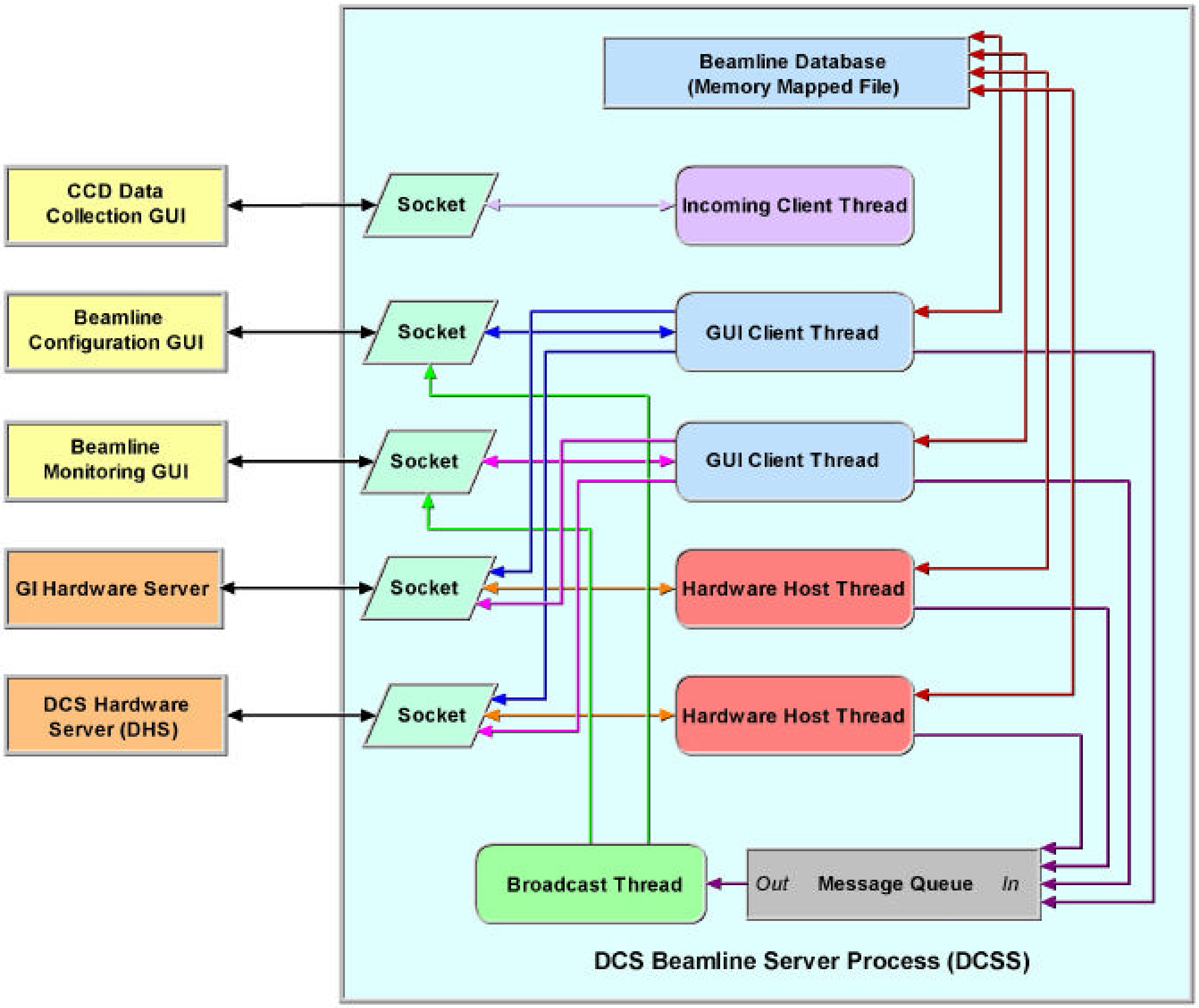
Message-Passing Architecture (Not Client/Server)

- Messages sent as simple text strings over TCP/IP.
- Messages are handled asynchronously.

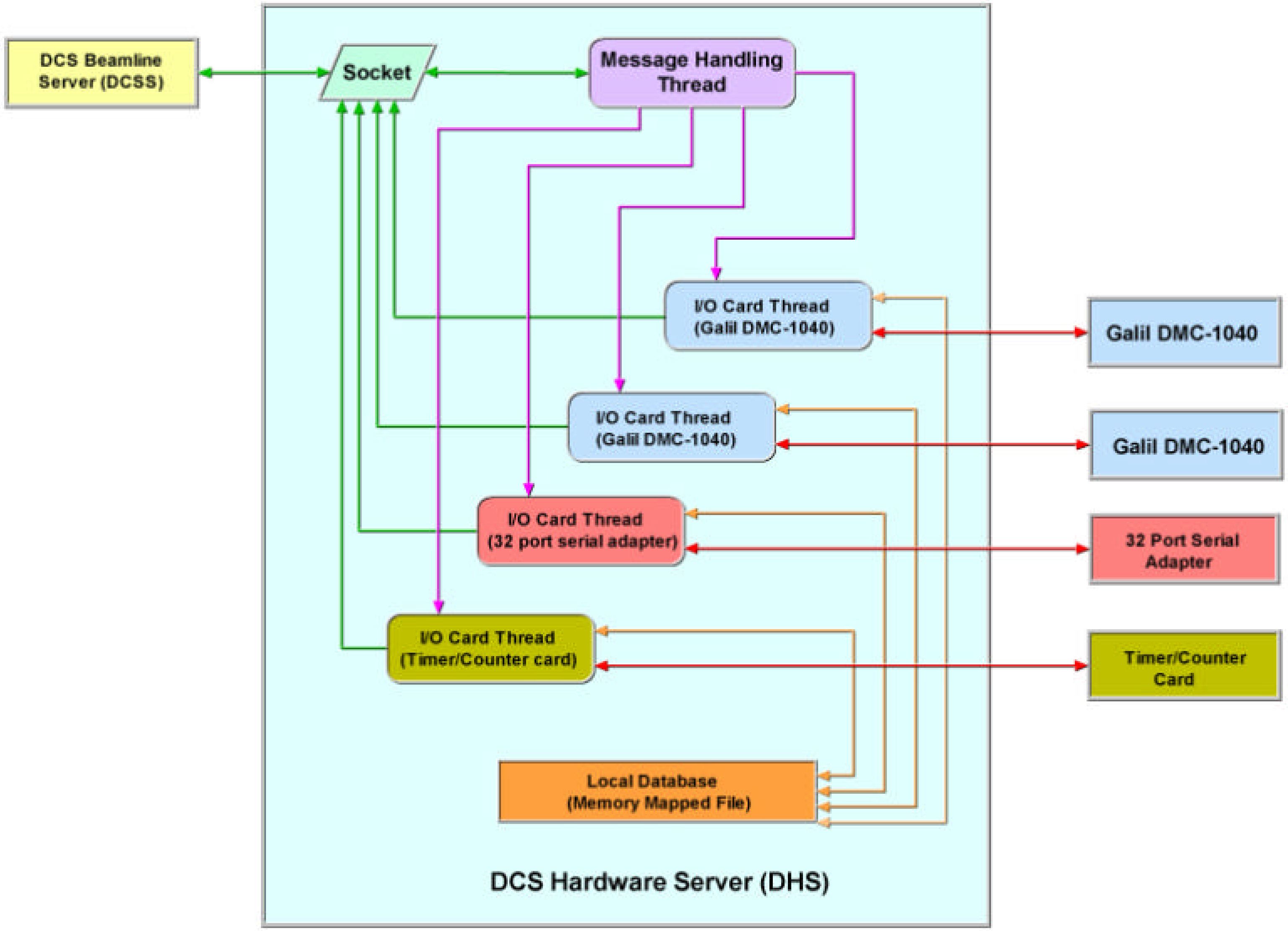
DCSS and DHS Written Using XOS Library

- Programs can run anywhre, on any platform.
- The ICS hardware server must run on VMS.

DCS Beamline Server (DCSS)



DCS Hardware Server (DHS)



General Beamline Control GUI