2020 NDIA GROUND VEHICLE SYSTEMS ENGINEERING and TECHNOLOGY SYMPOSIUM
VEA Technical Session
August 11-13, 2020 - Novi, Michigan

GVSETS 2020 Paper: Multiple Crew Station Xecutor (MCSX)

Shane G. Sopel, Mark G. Russell, Keith E. Zwick

Vehicle Electronics and Architecture (VEA),
U.S. Army Combat Capabilities Development Command (CCDC)
Ground Vehicle Systems Center (GVSC), Warren, MI

ABSTRACT

Technology and innovation are growing at a rapid rate, placing increasing demands on military vehicles. With these advances come additional burdens to our ground vehicle systems due to escalating threats in areas such as situational awareness and cybersecurity. In order to deal with this ever-changing threat environment, additional computing resources are needed. Given the additional costs of high performance hardware, harnesses, software development, sustainment, and licensing fees, consolidation of resources can be essential in reducing costs. Leveraging today's latest technologies in distributed systems, advanced microprocessors, and accelerated graphics, this research proposes a solution to consolidate multiple crew stations into a single processing resource. Not only are these computing resources more powerful, they come at a more affordable price when configured properly.

Citation: S. Sopel, M. Russell, K. Zwick, "Multiple Crew Station Xecutor (MCSX)", In Proceedings of the Ground Vehicle Systems Engineering and Technology Symposium (GVSETS), NDIA, Novi, MI, Aug. 11-13, 2020.

1. INTRODUCTION

Multiple Crew Station Xecutor (MCSX) is an innovative concept that allows for multiple independent crew stations to be simultaneously operated by using only one ruggedized computing unit. Soldiers seated at different crew stations will continue to seamlessly function mutually exclusively with each other, while still adhering to the configured soldier crew input and output (Crew I/O) required for their station's specific requirements. The hardware that makes up crew stations and the Crew I/O are a rugged display, a rugged computing unit with attached harnesses, and optionally a Keyboard Video Mouse (KVM) device. MCSX is a Linux based utility that leverages open source software packages which incur no additional costs. Using Linux provides the ability to deploy MCSX across multiple hardware architectures, such as ARM, x86-64, and Power PC. Furthermore, the multi crew station deployment highly leverages graphical hardware, software, and tools.

2. CREW STATION DISPLAY RENDERING

2.1 Rendering Software & Tools

The "Xecutor" in MCSX utilizes the Unix/Linux based windowing system ('X11', or simply 'X'), which executes all graphical display sessions. An X display Session is an instantiated GUI interface. Specifically, X11 is a Graphical User Interface (GUI) that renders windows and images. It also provides the framework and the tools necessary to allow a user to interact with mouse, keyboard, touchscreen, and other input devices. The individual session a single user has is known as a user session.

2.2 Display Rendering Architecture

X creates its user-to-display interfaces using a client-server software architecture model as described in [1]. The main purpose of the X server is to provide a graphical display and control of that display to users. Clients are the application(s) the user desires to display and interact with via the X window. Servers are numbered, starting with a 0 and prefixed with a colon. For example, when referencing the first server, command line users can map their client using the syntax DISPLAY = ":0". A typical dual display configuration is depicted below in Figure 1.

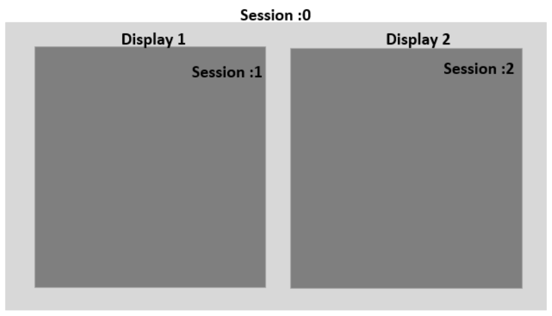

Figure 1 - Screen Layout

Using other X tools such as xorg-input-evdev, the X server is capable of handling user input such as mice, keyboards, or touchscreens. Figure 2 provides an example of a simple X server-client architecture.

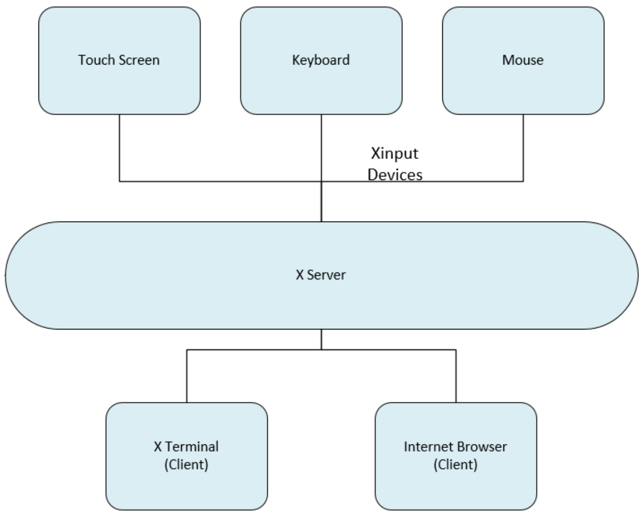

Figure 2 - X Window Server Architecture

2.3 Xephyr

X provides a large variety of tools and functions that allow developers to customize their environment. One of the tools used to develop MSCX is Xephyr. Xephyr is an open source software utility that implements X11 display server protocols. It extends and builds upon X by implementing X11 nesting, which deploys multiple software based X11 servers to crew station displays. MCSX deploys Xephyr at boot-up by initializing the parent X server, (server: 0) to spawn multiple child servers / sessions to span across multiple displays [2]. The parent server is responsible for performing all of the frame buffering (rendering display data on the physical screens) needed on each crew station display [3]. The beauty behind this configuration is users get infinite configurability of window screen placements, heights, and widths. An example of this can be observed below in figure 3.

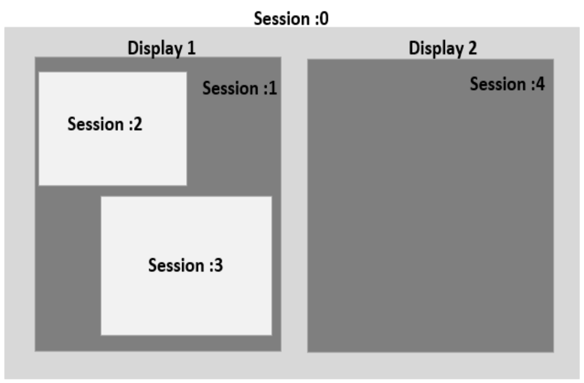

Figure 3. Xephyr Screen Layout

- Session :0 is the parent and is responsible for all resources and frame buffering for the entire graphics rendering real-estate.
- Session :1 is virtually connected to display one, allowing crew station A the capability to function independently.
- Session :2 and :3 are embedded into session :1; this is an additional feature operators can use for unique widgets and other apps executing on the hardware.
- Session :4 is virtually connected to display two, allowing crew station B the capability to function independently.

2.4 Mapping Input Devices

X11 and Xephyr are also responsible for managing crew input devices such as touch screen, bezel buttons, and keyboards. Xephyr requires mapping of physical devices to their respective X servers. In order to control each input device, every Xephyr session interacts directly with the underlying hardware device driver. Specifically, the Xephyr session takes control of the input device directly. This is accomplished through the use of Xephyr's evdev driver that communicates on behalf of the hardware device driver. Consequently, no other server will have access to these inputs once they are locked. The Linux bash shell input script to initialize hardware lock can been seen below:

```
  #!/bin/bash

  xkb='xkbrules=xorg,xkbmodel=evdev,xkblayout=us'

  #Keyboard Configuration
  kbd1='/dev/input/by-id/usb-
Dell_Dell_Multimedia_Pro_Keyboard-event-kbd'

  #Mouse Configuration
  mouse1='/dev/input/by-id/usb-
PixArt_USB_Optical_Mouse-event-mouse'

  #Start Crew Station Configuration
  startx -- /usr/bin/Xephyr -resizeable -origin +0+0
-fullscreen :1 -keybd "evdev,,device=$kbd1" -mouse
"evdev,,device=$mouse1" &
```

The script is used to initialize one independent crew station. Keyboard and mouse variables are added to set the device id paths needed for the hardware used by the specific crew station. The startx portion will start the xephyr executable and add the screen size requirements, the X server notation, and the locked in keyboard and mouse hardware configuration.

3. MULTI CREW STATION CONFIGURATION AND DEPLOYMENT

In the same manner as computing hardware for existing vehicular crew stations, there are common hardware interfaces required to operate a MSCX crew station. These interfaces consist of multiple dumb displays with bezel and touch screen capabilities, a single ruggedized computing unit that utilizes common hardware buses, and potentially one or more attached sensors or devices. Specifically, the dumb display is a hardware unit that has little to no processing capabilities. Additionally, it consists of a touch-capable surface that is populated with bezel buttons surrounding the outside of the screen. The ruggedized computing unit provides the crew with interfaces that utilize common communication hardware buses. These interfaces consist of Controller Area Network (CAN) bus, Universal Serial Bus (USB), Ethernet, Graphical Processing Units (GPUs). These interfaces need to be capable of providing video, diagnostic, audio and network interfaces.

3.1 Configuring Resources for MCSX

One of the benefits behind MCSX lies in its ability to deploy multiple software applications across multiple crew stations while executing on a single system. The utilization of Xephyr provides the environment in which applications can execute independently of each other, while sharing the underlying computing resources of the host system. Along with graphics processing, the ability of the system to share common computing resources is crucial to MCSX and should also be transparent to applications attached to each crew station. Each software application should behave as if it is operating on a standalone system. All that is required is to configure the applications to work with the various protocols and exposed I/O devices of the underlying host system. This flexibility of the divvying up of resources amongst the crew stations and the sharing of different communication protocols is crucial to its success. Examples of shared resources are:

- **Graphics Processing Units (GPUs):** Crew Stations tailored for MCSX may have one or more GPUs. A concept known as 'multi-seating' exists for dual GPU configurations, but comes with greater challenges, specifically with configuration of user accounts. Additionally, deploying crew station computing hardware with more than one GPU presents added challenges that include managing heat generation and dissipation, increased costs, and likely limiting potential vendors. The ability to drive multiple mutually exclusive displays and crew stations with one GPU can be facilitated by the use of Xephyr, as described above. This software enables user sessions to create the same interface as the multiple GPUs' sessions, without having the same concerns of power consumption and heat dissipation that multiple GPUs have.

- **Network Interface Card (NIC)**: A requirement of having multiple crew stations may be the need to have independent IP addresses for each station. If a requirement of a physical adaptor to each IP is not needed, IP aliasing or routing can be used. According to The Linux Foundation, IP aliasing [5] or IP routing [6] allows for the binding of multiple or many virtual interfaces (aliases) to one physical card. These IP addresses can be part of the same subnet or on two different logical network subnets. Therefore, each crew station can be configured to work with an independent IP address while providing it a unique address on the network.

- **Controller Area Network (CAN Bus):** CAN data messages can be shared between each application running on a crew station through use of CAN software, preferably SocketCAN. Unlike traditional CAN software which typically talks to a character device similar to a serial driver and has limited functionality, The Linux Foundation states [4] SocketCan provides a socket interface which builds upon the Linux network layer overcoming the limitation of one process reading CAN message data. Communication with the CAN bus is done analogously to the use of the Internet Protocol. Each application can have its own CAN Id for communicating and use one CAN interface for reading and writing on the CAN Bus. If CAN bus data is to be restricted from all applications, another application layer outside of the Xephyr X11 server could be added to read in all CAN communication. It can then act as a CAN bridge providing applications with information via Internet Protocols or other network communication mechanisms.

- **Audio**: Due to the complexities of the multiple crew stations sending/receiving audio at the same time, a mechanism may need to be added to broker the audio produced by each crew station or application. If multiple processes attempt to use a sound card simultaneously, it is necessary to broker the audio with a server like PulseAudio. PulseAudio is described by Archlinux [7] as a sound server that runs as middleware between your applications and hardware devices. A sound server would be set up as a background process accepting sound input from various sources and sharing it with the other processes running in the various crew stations.

- **Serial Devices**: Due to the limitations of only one process reading a specific serial device, it would require a publish-subscribe methodology [8] where one process running in a crew station, or the host system itself, would read the data and share it to the other processes running in the various crew stations.

Through the use of Xephyr, and the shared system's resources Xephyr is running on, the possible application configurations are only limited by the limitations of the system resources itself.

3.2 MCSX Computing Overhead

When deploying MCSX, it's important to be cognizant of the additional overhead that may be incurred as a result of running multiple software applications from one computing unit. This research recorded some basic metrics that track the use of additional MCSX resource sharing. The data was obtained by exercising VEA software that was executed on a 2nd generation dual-core 2.2Ghz Intel Core i7 with an onboard Intel graphics processor. The following metrics were recorded when measuring some of the previous section's additional capabilities:

- **Shared CAN Bus Monitoring.** The demo CAN monitoring software instantiated twelve CAN-bus Line-Replaceable Units (LRU) proxy stacks. Each proxy monitors for LRU-specific CAN traffic and decodes incoming messages at a rate of 1000 Hertz. Mimicking a non-MCSX deployment, only an average of 10% of one CPU was utilized throughout the run. When a second instance of this software was launched (which implies MCSX is configured to simultaneously execute another similar crew station application), the average CPU utilization doubled to 20% of one CPU as expected.

- **GPU Utilization.** The Linux command-line Intel GPU tool was launched to measure the overhead incurred when running VEA's event and QT-based graphical user interface application. Only 3% GPU usage was measured when rendering one crew station application. The GPU usage was upped to 7% when deploying Xephyr and two simultaneous crew station graphical user interface applications.

- **RS232 -> Ethernet Publish / Subscribe.** The example application consumes a twenty four byte diagnostic message at a rate of 1 Hz. Reading RS-232 data at 9600 bps incurs a small amount of mandatory overhead, but the diagnostic data still needs to be distributed to multiple crew stations. In this test configuration, there is a UDP publisher application and there are multiple UDP subscriber applications. Each application was extremely light weight as they all reported an average of 0% CPU usage.

Modern CPUs and GPUs are well optimized and are more than capable of efficiently sharing resources. Physical devices such as network cards are capable at operating at speeds of up to 2 Gbps; allocating this amongst multiple crew stations provides much more bandwidth then most LRUs require. These findings support the notion that the MCSX architecture will not add many constrains in deployment.

4.0 REFERENCE ARCHITECTURE

An example of an architecture leveraging MCSX is depicted in the Figure 4 reference diagram. Inside a vehicle there are two ruggedized systems running MCSX on two different network enclaves. The network consists of an unsecure enclave and a secure enclave to provide protection of sensitive information from being accessed by unauthorized users or applications. This proposed design will leverage MCSX to minimize space claim and costs in a constrained system.

Computer Systems: Three crew stations and one targeting system are running within two ruggedized computer systems. Operator two has the ability to switch to another system in their existing space.

Displays: Four displays were needed for this system but with the use of the KVM using the shared resources of the ruggedized computer in the unsecured enclave, three were sufficient.

Shared Resources: Through the use of shared resources such as DVI, HDMI, USB, and Network Interface Cards, resources can be configured and routed to each crew station. This will allow for each crew station to communicate independent of each other.

Cross Domain Data: The targeting system running on the ruggedized computer with crew station C is able to share data with crew station C, while simultaneously having its video sent to the unsecured enclave via HDMI to the KVM. The KVM would then display that video to operator 2 without any of it being accessed by crew station B applications.

By leveraging MCSX, this reference architecture effectively reduced the need of four screens to three and the use of four independent computer systems to two, which collectively operates four applications. With the ability of utilizing shared resources on two ruggedized computer systems, MCSX has been able to save space by reducing in half the amount of hardware and cabling, and the potential reduction of a fourth operator. With this reduction of space comes a reduction in cost. Integrating this architecture into a fleet of vehicles could provide significant savings for the United States Army.

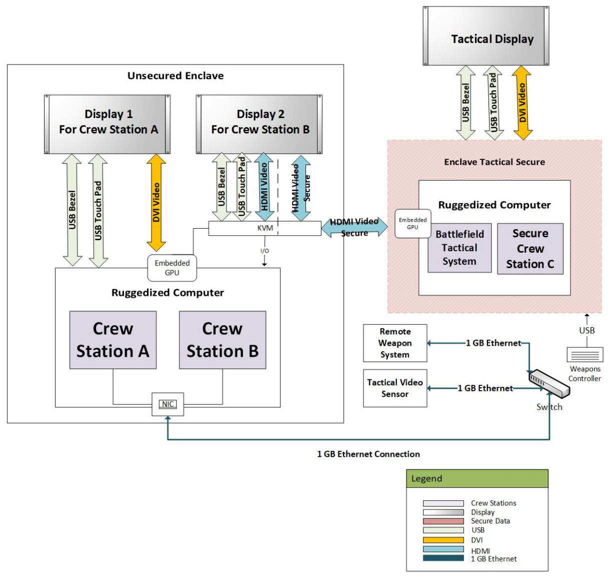

Figure 4. Reference Architecture

5. FIELDING & COST IMPLICATIONS

Vehicle platform engineers and program managers have difficult hardware and software procurement tradeoffs to make when developing and fielding the vehicle of tomorrow. If total vehicular Size, Weight, Power, and Cost (SWaP-C) drive the majority of the system requirements, implementing MCSX provides savings on all of the aforementioned measurables. Naturally, if a vehicle can deploy with the same functionality and one less computing crew station, total cost, power consumption, weight, and space claim can be reduced. Furthermore, the number of vehicle harnesses that are required will be reduced as well.

5.1 SWaP-c Assumptions

Given costs for average crew station hardware items, small and large vehicle harnesses, and software runtime / deployment software licenses at large economies of scale, the potential projected cost savings for a crew station in a fleet vehicle can be assumed:

Hardware Item	Cost
Ruggedized Computing Unit	$6,000.00
Ruggedized Dumb Display	$10,000.00
2 Smaller Harnesses	$1500.00
1 Large Harness	$2000.00
Software Support Fees	$1000.00
Total	**$20,500.00**

Table 1: Hardware Item Costs

Using the example reference architecture provided above in the previous section, this research suggests that a vehicle can consolidate four crew stations into three. With the average projected cost of $20,500 per crew station, but with the addition of one KVM, the following per-vehicle costs with and without MCSX can be extrapolated:

Cost	With MCSX	Without MCSX
Total Crew Stations	(3 @ $20,500): $61,500	(4 @ $20,500): $82,000
KVM	$3,500	-
Total	**$65,000**	**$82,000**

Table 2: Per Vehicle Costs

The Stryker vehicles provide an excellent example of projecting cost savings across the breadth of an entire fleet. Given that there are approximately 3000 fielded Strykers at any given time, and, again this research approximates that the average cost savings per vehicle is $20,500, then the program management office could potentially realize a cost savings of 3000 x $20,500 = $61,500,000. In addition to saving costs, other tangible savings can be quickly achieved by deploying an architecture that uses MCSX, including:

- **Logistics**. Vehicle logisticians will benefit by having to track one less crew station's worth of part numbers.
- **Size**. An estimated 1-2 cubic feet of value real estate would be saved, as well as improved ingress / egress with reduced harness footprints.
- **Weight**: Not significant, as it may only save a maximum of ~20-40 pounds.
- **Power**: Approximate savings of ~30-60 max watts.

6.0 SUMMARY

The Multi Crew Station Xecutor provides unique methods to consolidate crew stations, hardware resources, and software applications. This proposed solution provides opportunities for vehicle platform designers to save cost and space where multiple crew stations are combined into one computing unit and one display while simultaneously arbitrating between connected crew input devices. These advantages for procurement and deployment must be carefully weighed against the inherent risks; in the event of a computing hardware failure, multiple crew stations may now become inoperable. Nevertheless, simpler backup solutions would likely mitigate some risk. Through effective leveraging of shared hardware computing resources as well as designing properly configured and managed software applications, MCSX has the potential to change the landscape of vehicular hardware deployment and offer the possibility of saving millions of taxpayer dollars.

References

[1] The X Window System: A Brief Introduction
http://www.linfo.org/x.html

[2] Xorg Xephyr
https://www.x.org/archive/X11R7.5/doc/man/man1/Xephyr.1.html

[3] Xorg Xvfb
https://www.x.org/releases/X11R7.6/doc/man/man1/Xvfb.1.xhtml

[4] SocketCAN
https://www.kernel.org/doc/Documentation/networking/can.txt

[5] IP Aliasing
https://www.kernel.org/doc/html/latest/networking/alias.html

[6] IP Routing
https://wiki.linuxfoundation.org/networking/iproute2

[7] PulseAudio
https://wiki.archlinux.org/index.php/PulseAudio

[8] Publish-subscribe pattern
https://en.wikipedia.org/wiki/Publish-subscribe_pattern