High-Level Graphical User Interface to Streamline Mission Management of Dynamically Growing Data Transport Systems

Richard W. Hoffman III

GDP Space Systems

ABSTRACT

As data transport systems become exponentially larger and more complex, the need to simplify the level of user involvement in establishing the intercommunication pathways becomes increasingly vital to streamlined, effective mission management. The proliferation of open-architecture, modular approaches to data transport and multiplexing systems shows the need for a unified, high-level control scheme that helps to flatten the users' learning curve for increasingly sophisticated, expanding systems. Implementing a control package with the functionality described in this paper will improve the user experience by eliminating the need for low level hardware management, minimizing system network footprint and unifying this functionality for a diverse hardware package.

KEY WORDS

GUI, Network Appliance, Object Oriented Programming

1. MISSION MANAGEMENT OVERVIEW

As missions become increasingly sophisticated, so too does the data transport framework required to sustain and manage them. With an increasingly complex, dynamically growing acquisition and transport system, the task of maintaining control stands to become unwieldy. The feat of managing these control packages has become one that requires a multi-discipline background, incorporating the tasks of IT network management, information assurance (IA), and communications engineering. The oft-overlooked role of the management software is one that can either greatly aid in this sometimes herculean effort, or gravely hinder it.

In addition to an increase in the number of mission critical data sources, many of those sources are increasingly capable of generating data streams at very high rates. Monitoring

and management of system node level aggregate bit rates is also of concern where many of these potentially high bit rate sources must shared limited, though seemingly extensive bandwidth resources.

In many mission scenarios there is a need for rapid, on-the-fly reconfiguration, whether it is to accommodate a repurposing of resources, or to facilitate the implementation of a workaround for a failed module. When the magnitude of the sensor suites for these missions is as extensive as those in many current and forecast systems, there is no reasonable way to accommodate a large-scale reconfiguration of those resources without an adequate level of automation. Automating these "mission switches" can itself prove to be a complicated process, but with attention and forethought paid to the management software, the urgency is placed on the implementation of a solution rather than the identification of the problem.

With any sensitive mission, timing is critical whether it is pre-launch configuration of the mission, the monitoring of a live mission, or the playback of a previous event. Provisioning for these time-critical hurdles enables the user of a high-level interface to manage these highly dynamic systems with very low response times.

2. SYSTEM MANAGEMENT IMPLEMENTATION

There are multitudes of methods an engineering team can opt to utilize when addressing the issue of managing these types of dynamically changing systems. In an attempt to accurately and fairly inform the user, some of the most common forms of management configurations are described herein. Each of these configurations assumes that all network appliances in question can communicate amongst themselves.

Standalone, Network Independent

> These systems are best described as those in which one or more devices with a network presence are capable of transporting data from one to another. Each of the networked appliances requires direct input from a user with *a priori* knowledge of the device's mission application, its network footprint location, pre-apportioned bandwidth, and inter-device path, as well as data manipulation parameters. Systems utilizing this configuration are configured as determined by the network administrator and communications engineers. There is no single control package for these units and each may utilize its own proprietary configuration scheme. This is the most unwieldy configuration for systems that are intended to grow and change as mission variables are modified.

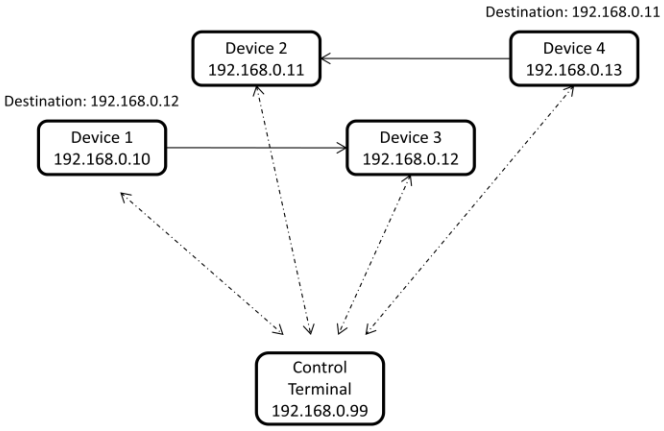

FIGURE 1: Standalone Network Configuration

Encapsulated Standalone

An encapsulated standalone system is one in which the individual network nodes are themselves unaware of any other compatible network presences. These systems can be wrapped with a software package that addresses these systems from a single point of control while still allowing the systems to function independently, without any mission context for their operation. Systems configured in this way typically make use of some network-level management interface such as SNMP to retrieve status and affect configuration changes. These systems enjoy some of the benefit of network awareness without the ability to interact directly.

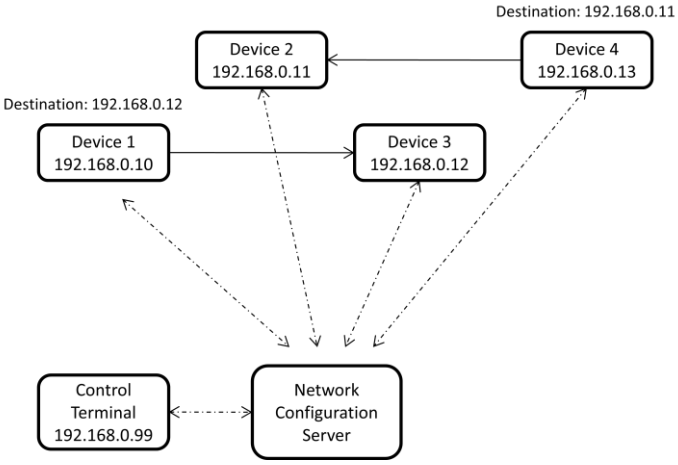

FIGURE 2: Encapsulated Standalone Network Configuration

Network Omniscient

This configuration is capable of a higher degree of autonomy from the user than the previous two. Each networked node can control itself and any path-able, compatible

remote nodes. On top of this ability to control other units, each unit is itself aware of the status of the remote nodes, and can react almost instantly to situations ranging from loss of power on a remote unit to runtime altered hardware configuration with a variety of user specified behaviors, defined prior to mission launch. These systems, while complex, enable a user to accomplish a large number of setup, monitoring, and termination tasks without a need for direct, time-sensitive, user input.

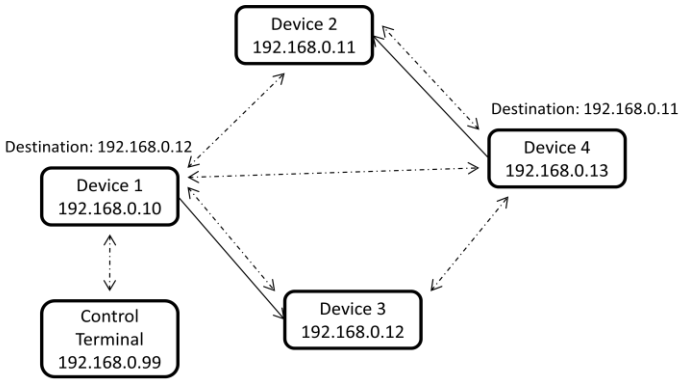

FIGURE 3: Omniscient Network Configuration

In either of the two standalone systems detailed above, the configuration of the network presence of each of the individual modules is itself a large task.

The process of managing this many devices on a network requires the user to make the appropriate management connections to each of the network devices individually, and then coordinate network addressing based upon the mission parameters. Additionally, this means of system configuration adds a degree of ambiguity into mission management in that there are no implicit channels through which endpoint status can be relayed from one network node to another. The user is then forced into acting as the sole provider of real-time mission reconfiguration.

In contrast to the stand-alone system configurations, the omniscient configuration relies on each network node acting not only on its own ends of a set of connections, but also on the rest of the system nodes, passing configuration messages and notifications between them to facilitate complete inter-chassis system status awareness. As implied by FIGURE 3 above, any network node still network path-able can react to and report any dynamic event from any other node. Though there are several ways to facilitate this system-wide, event driven philosophy, the emergence of data synchronization techniques like database clustering help to eliminate the need for a central server appliance to manage these events, such as would be required to implement similar functionality within an established framework like SNMP.

3. OBJECT ORIENTED PROGRAMMING APPROACH

In order to leverage the immense power of the object oriented programming (OOP) philosophy, it's critical that a system designer develop comprehensive, extensible, inheritable data structures to represent the critical elements of the system. Critical system elements that often require special OOP consideration include the device chassis, the modules, module ports, the chassis links, and the data connections themselves.

Attention must be paid to the process of data structure abstraction and interface definition in order to ensure that all elements can be properly hierarchically nested. This consideration helps to establish the relationships between elements and makes more complex behaviors easier to define. An example of one such abstraction scheme can be understood from the diagram and code fragment given in FIGURE 4 and FIGURE 5 respectively.

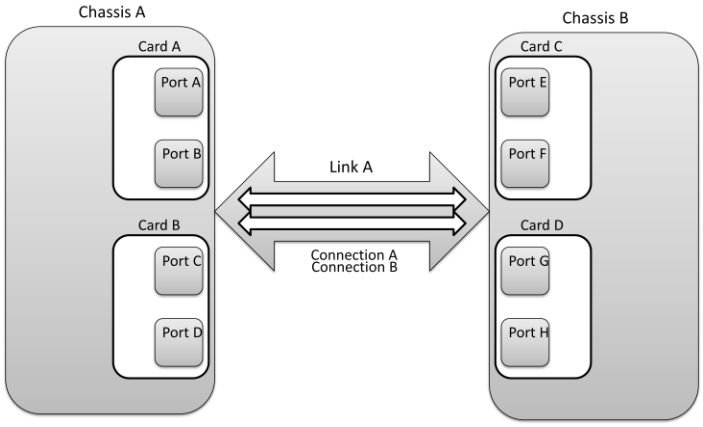

FIGURE 4: System Hardware Configuration Diagram

```
Class GdpChassis
{
  List<GdpCard> _childCards;
  ...
}

Class GdpCard
{
  List<GdpPort> _childPorts;
  GdpChassis _parent;
  ...
}

Class GdpPort
{
  GdpCard _parent;
  ...
}

Class GdpLink
{
  ...
}

Class GdpConnection
{
  GdpLink _parent;
  ...
}
```

FIGURE 5: Abstraction Pseudo-Code

Relying heavily on the *encapsulation* principle of OOP, the system designer is now able to present an interface in which the user is no longer concerned with specifying device addresses or connection destination addresses because they are encapsulated within the

abstract system objects defined as shown above. In FIGURE 6 below, an example of creating a connection between two ports, within separate cards, which are themselves in separate chassis linked via ethernet is shown. There is no need for the user to specify destination addresses, ports, which data link to use, or optimal bandwidth and network QoS settings.

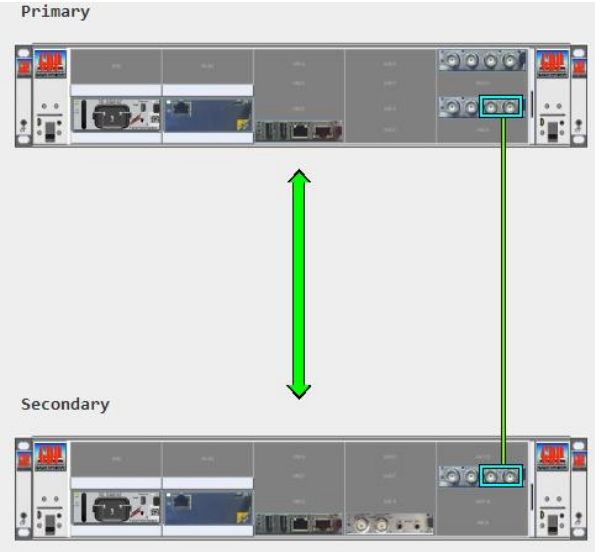

FIGURE 6: Abstracted Data Connection Creation

The extension of these data structure objects into visual elements for use in the GUI is fairly straightforward. With the appropriate choice of OOP development language, constructs such as Java and C# Interfaces can be utilized to ensure that a basic set of behaviors is handled in every element of the GUI. The benefit of this is realized when considering a changing, reconfigurable system. Assuming that the hierarchical relationships are established by the designer, reconfiguration behavior can be expected to have been defined in the interfaces associated with the affected system elements, whether they are high level entities like device chassis, or lower level entities such as device ports.

It is apparent that on a sixteen card, thirty-two port device, for example, the process of reacting to a module failure could involve a substantial amount of guess work, re-cabling, and user interface activity. However, with a system wherein the behavior for a failed module is defined to automatically switch to another available port, downtime on the data stream could be milliseconds as the system automatically reconnects.

In the above example, the entirety of the task for the user is to switch appropriate cabling, if necessary, to the newly activated port. The user interface would be expected to notify the user of this action through a dialog box or prompting message. An additional benefit is gained from the fact that the question of which devices and modules are affected by the failure can be displayed in a foolproof way such as in FIGURE 7.

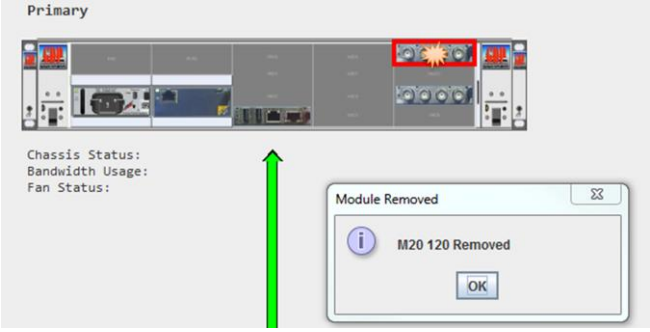

FIGURE 7: System Event Notification

An additional benefit of this methodology is one that comes directly from the principle of *extensibility*. Systems built upon complex, relational data structures could be expected to be cumbersome to update for use with future extensions, such as new I/O modules. By properly utilizing object inheritance and a defined software interface, the addition of a new module could be realized with nothing more than the implementation of the required extension and interface methods, as demonstrated in FIGURE 8.

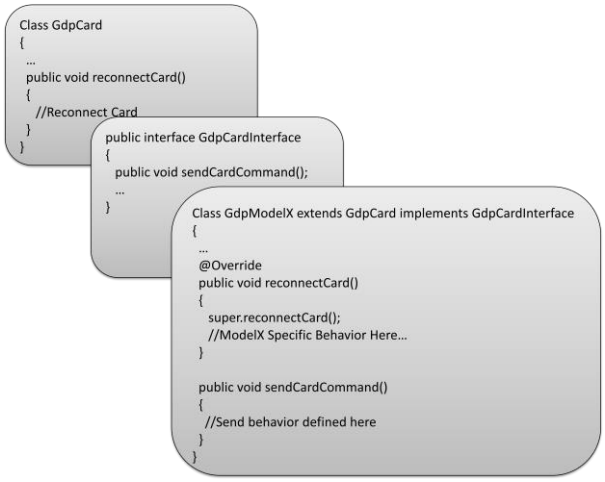

FIGURE 8: Inheritance/Extensibility Pseudo-Code

4. GRAPHICAL USER INTERFACE

Within the commercial sector, advanced graphical interfaces have become an essential piece of any product. These interfaces feature intuitive command contexts to facilitate an ease-of-use that non-commercial interests have not previously enjoyed. Utilizing these same design principles within the high stakes, dynamically changing environment in which the telemetry community operates offers an opportunity to simplify complex, unwieldy, oftentimes legacy processes and present relevant information at a single point of access for

at-a-glance use by the system operator. This governing design principle mandates that all pertinent system information be readily accessible without requiring a user to navigate away to a view of a device without a context for its function within the wider system as shown in an example in FIGURE 9.

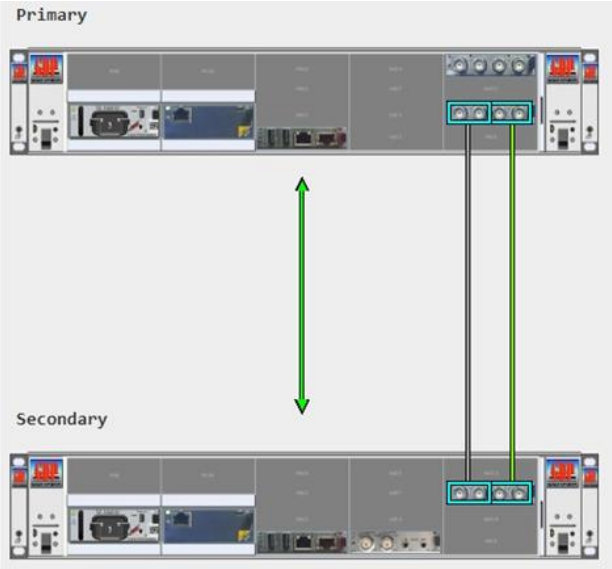

FIGURE 9: System Level GUI

The minimal primary display of a system with this type of interface should be a macro-view of the whole system, including every networked device, its modules (if it contains any) and all inter-device data paths. Additionally, users must be able to access information from more focused levels of the system, such as how much of the available bandwidth of a single device is being used by its data connections, or what the device operating temperature currently is.

In addition to the system-wide view offered by the minimal dataset detailed above, the user is also offered a window into the operation of devices communicating in an inter-device configuration, and in the case of a system where devices may contain several functional modules, into the operation of these intra-device configurations. Having functional access to and control of every level of a system and its children devices from a single view allows the user to determine at which level to handle varied tasks, such as data path disruptions or device failure.

Understanding that the overarching design principle is that more system data is represented in a visual manner, important system information is displayed in visually sophisticated, easily deciphered display elements such as the aggregate device bandwidth usage chart or system fan tachometer displayed in FIGURE 10 below.

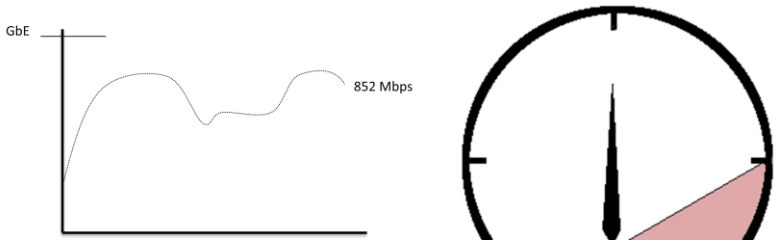

FIGURE 10: Visual System Status Elements

The same methodology can be applied to configuration of network devices, their data channels, and their data stream connections. In configurations where devices may contain one or more distinct data acquisition/transfer modules, the same display and configuration theory is applied, offering context sensitive controls determined by the function of those modules in the wider system configuration.

5. CONCLUSION

The proper implementation of a high level, graphical user interface can serve to greatly reduce the load on a user before, during and after a mission. Removing the need for a user to maintain a comprehensive knowledge of network configuration, device configuration parameters, or even device mission usage improves total mission up-time, even during difficult to anticipate scenarios.

A design following the above principles, while more sophisticated and difficult to implement within existing network frameworks, affords immense benefits to the user and enables a wider array of users to participate in the mission management task.

6. ACKNOWLEDGEMENTS

I'd like to acknowledge my wife, Amanda, who patiently listened as I sounded out all of the "tough stuff".

7. REFERENCES

Johnson, Jeff, Designing With The Mind In Mind, First Edition, Morgan Kaufmann, Burlington, MA, June 3 2010