Space plug-and-play Avionics in the Naiad project[*]

Christoffer Holmstedt
christoffer.holmstedt@gmail.com

ABSTRACT

Space plug-and-play Avionics technology can easily be adapted to other fields such as autonomous underwater vehicles. During the ”Naiad project” an initial attempt was made to do this. This report goes through the basics of Space plug-and-play Avionics, the derivative work of ”Virtual Plug-and-Play Network” and the problems experienced throughout the project.

In the end the problems were too big to solve during the project so support for Space plug-and-play Technology was dropped. Conclusions presented focuses mainly on future work.

1. INTRODUCTION

As a requirement for the project the autonomous underwater vehicle were to use Space plug-and-play Avionics technology to make it modular in the sense that it should be easy to add and remove parts from it for different missions. In this report lessons learned from this work will be presented for future interest in this area.

1.1 Space plug-and-play Avionics

1.1.1 Background

Space plug-and-play Avionics (SPA) has been developed by NASA and partners to improve the time it takes to assemble a new satellite for specific missions. In other words the goal is not to decrease the development time of specific components but instead design a plug-and-play system such that no matter what the mission requirements are, NASA can put together a sattelite within hours or days that meet a new mission's requirements.

[*]This report was written during the fall of 2013 in an advanced level project course at Mälardalen University, Sweden.

1.1.2 The Standard

SPA is specified in a set of standards documents, each standard specifying functionality in a specific area ranging from ”SPA Logical Interface Standard” to ”28V Power Service”. The standard documents define everything from power supply levels to how components should communicate with each other. Focus during the Naiad project has been on a subset of all the standards more specifically the following ones:

- Local Subnet Adaptation Draft
- Logical Interface Standard [4]
- Networking Standard [5]
- SpaceWire Subnet Adaptation Standard [6]
- Mini-PnP / SPA-1 Protocol Draft

The Local Subnet Adaptation Draft defines how inter-processing communication should be done. It defines the use of UDP/IP as the main protocols to use. The Logical Interface Standard defines how hardware and software components should communicate on an application level. The Networking Standard, SpaceWire Subnet Adaptation Standard and the Mini-PnP / SPA-1 Protocol Draft all defines how the respective subnets should work and be connected with each other.

The network standard defines how enumeration should be done for all components in the network while the hardware specific adaptations defines how adress resolution should be done in respective hardware specific network.

1.2 SPA over CAN Bus

Another requirement from the customer was the use of CAN Bus [2] for communication between sensors and actuators. This meant that as a main part of the projects SPA implementation was to design and develop SPA functionality over the CAN Bus. No previous work was found in this field.

1.3 SPA with Ada

A third requirement from the customer was the use of Ada as the programming language. Previous implementation of the SPA standard exists in C/C++ but not in Ada.

1.4 Virtual Plug-and-Play Network

Virtual Network Protocol (VNP) [7] or Virtual Plug-and-Play Network, VPPN (working titles) is an initiative to bring SPA technology to the consumer and business market outside of the space industry. VPPN puts focus on the software part of the SPA standard and have therefore removes all parts related to hardware specifics such as which hardware connectors to use. To clarify this, SPA and VPPN is as of writing names of the same technology but with different business targets. Throughout this report VPPN will be used except when refering to specific reports.

The following parts of the report are structured as follows. Section 2 goes through what has been done with VPPN in the Naiad project and section 3 focuses primarly on future work.

2. IMPLEMENTATION

2.1 Virtual Plug-and-Play Network

2.1.1 Design

The goal with the first implementation was to test the capabilities of VPPN. This included basic infrastructure such as a Lookup Service, a CAN bus gateway, an application (a consumer of data) and a service provider. Figure 1 shows the basic components needed and how they are interconnected in suggested minimal implementation.

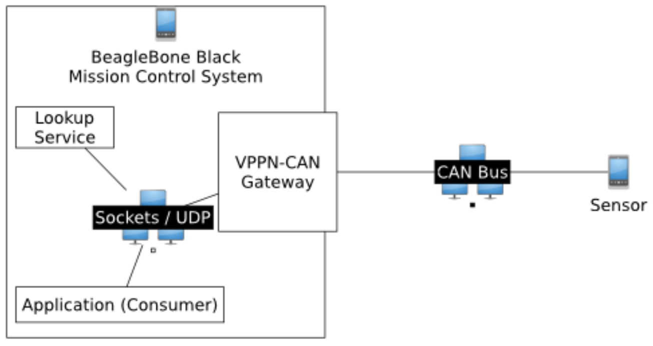

Figure 1: Overview of the minimal VPPN design with CAN bus Gateway.

Each component's responsibility is described in the following list and an example connecting all components together is presented after that.

- *Lookup Service (LS)* keeps track of all connected components respective interfaces (available and required data).
- *Application (consumer)* uses data supplied by the service provider.
- *Service Provider* could be a sensor of some kind that applications can subsribe to for reguarly updates of the sensor value.
- *VPPN-CAN Gateway* routes traffic from the local subnet in the processing unit to the CAN bus local subnet.

In a fully operational VPPN network adress resolution and initial configuration is partly done by other components not used in this minimal design, for the minimal implementation hardcoded settings were planned instead.

After initial configuration each component sends information to the Lookup Service when requested in the format of "Extensible Transducer Electronic Data Sheets" (xTEDS) which is a XML file with a predefined structure. This file includes respective components interfaces and must be created when the component is constructed (it should be stored together with the component it describes).

The Lookup Service then goes through all xTEDS to find out which components require information from other components. The Lookup Service in the minimal implementation would find that the "Application" requires sensor information of the same type the "Service Provider" supply. The Lookup Service would then report back to the Application with information that the Application should contact the Service Provider. The Application would then ask for required data or subscribe to the Service Provider's sensor information in a peer to peer manner.

Whenever the Service Provider on the CAN bus subnet communicates with another component outside of the CAN bus subnet the VPPN-CAN Gateway routes the traffic. To be able to do this routing, the VPPN-CAN Gateway is also responsible for the adress resolution in the boot up phase for the CAN bus subnet.

2.1.2 VPPN and CAN bus

In the minimal design the VPPN-CAN Gateway component runs on the same processing unit as the Mission Control System and connects with other components on that local subnet through UDP/IP through the loopback interface.

In a VPPN network each component, both hardware and software, must have a unique logical address. When starting the system the responsiblity of the VPPN-CAN Gateway is to enumerate the CAN bus for all connected components and assign a local subnet address to each one. This contradicts the behaviour of the CAN bus which is message based and not address based. As per request from the customer the project went forward with the minimal implementation design without putting any attention to the contradiction.

The main issue to solve is the assignment of addresses. In IP networks and VPPN Local Subnets a common address is used to make the network manager aware of the newly connected component. For IP networks newly connected network devices send a broadcast message to all other devices on the same local network with it's own address as 0.0.0.0 (IPv4). The DHCP server then replies with a response that includes a new IP address the device can use. In the SPA Local Subnet Adaptation Draft newly started processes sends a message to the loopback interface at port 3500 requesting an address. Together with the request for a new address a unique identifier is included in the message (e.g. MAC Address for IPv4 networks).

Applying the same behaviour to the CAN bus would cause the CAN bus to malfunction due to collision errors when

Table 1: Mapping of message ID for address based communication over CAN Bus.

Prioritisation	Msg	Sender Address	Dest. Address
10 bits	3 bits	8 bits	8 bits

multiple CAN hosts transmit the same message ID but with different payloads (including the unique identifier). This is related to the bootstraping problem others have tried to solve with ”IP over CAN” [1, 3].

For the minimal implementation the project went forward with inspiration from Ditze et. al [3] where the 29 bits message ID from the CAN bus 2.0B specification is divided into different subparts. It starts with a 10 bits prioritisation band, 3 bits message type field, 8 bits sender address and the last 8 bits represent destination address. This can be seen in table 1.

To eliminate the problem with collisions on the CAN bus unique component identifiers must be used within the message id. One solution explored was to define a message type as ”DHCP” request and use both the sender and destination address as a 16 bits unique data field. 16 bits is not enough for MAC addresses used in IPv4 devices (48 bits), unique identifiers in IPv6 devices (64 bits) nor the 128 bits CUUID used within VPPN. At this point no further work took place regarding this topic.

2.1.3 VPPN and Ada

During the initial implementation phase focus was put on Ada functionality and getting the communication between software components working. UDP/IP and TCP/IP were tested and it was clear early on that UDP/IP wouldn't work very well so the project continued with testing TCP/IP together with Ada Tasks more thoroughly. It was during a meeting with the customer at this point in the project the customer made clear that sockets were a no-go. TCP/IP communication was dropped in favour of ”Protected Objects” from the Ada language for inter-process communication.

At this point it was decided to drop VPPN from the project, prioritising other requirements from the customer.

3. CONCLUSION

Due to the limitations presented in this report Space plug-and-play Avionics (SPA) have not been used in the final AUV. The following two sections show how to continue with the combination of SPA/VPPN, CAN Bus and Ada.

3.1 Virtual Plug-and-Play Network

3.1.1 VPPN and CAN

In section 2.1.2 the problem with too few bits available to work with when it comes to address resolution was presented. Alternative solutions not explored could be to store a part of the unique addresses in the CAN Bus Message ID and the rest in the payload. A second alternative would be to use a randomly generated identifier and with math prove the unlikelihood of a CAN Bus collission.

Suggested work for the future is to tackle the problem starting from the now expired IETF draft presented by Cach and Fiedler [1]. The benefit of this solution is a smaller prioritisation field which make it possible to use more bits for addressing during the address resolution phase. Although it is worth remembering that the solution presented by Cach and Fiedler is not designed for systems with hard real-time requirements.

3.1.2 VPPN and Ada

The main lesson learned during development of the minimal implementation presented in this report was the lack of VPPN infrastructure components to test with. Most notable the Central Addressing Service (CAS) and the Lookup Service (LS). If UDP/IP were used, previous implementation in C/C++ could be used as drivers/stubs while developing the needed infrastructure in Ada. As that is not the case the most reasonable approach to implement VPPN in Ada would be to start with one local subnet within one single processing unit.

The first step would be to implement the CAS, the LS and the Local Subnet Manager (SM-L). After that a test service provider and test application could be implemented within the same subnet before moving on with VPPN Gateways to other subnets.

The reason for implementing the CAS, LS and SM-L in one go is that without CAS the SM-L can't give out new addresses on the local subnet and without the SM-L the CAS can't be connected to the local subnet (without hardcoding address information). The LS is then needed before adding applications and service providers so the VPPN network can handle actual data.

4. REFERENCES

[1] P. Cach and P. Fiedler. Ietf draft - ip over can. `http://datatracker.ietf.org/doc/draft-cafi-can-ip/`. Accessed 2013-12-02.

[2] Can bus 2.0 specification. `http://www.bosch-semiconductors.de/media/pdf_1/canliteratur/can2spec.pdf`. Accessed 2013-12-02.

[3] M. Ditze, R. Bernhardi, G. Kämper, and P. Altenband. Porting the internet protocol to the controller area network. `http://www.hurray.isep.ipp.pt/rtlia2003/full_papers/8_rtlia.pdf`. Accessed 2013-12-03.

[4] Space plug-and-play architecture: Logical interface (aiaa s-133-3-2013e). `http://arc.aiaa.org/doi/book/10.2514/4.102318`. AIAA Standard.

[5] Space plug-and-play architecture: Networking (aiaa s-133-2-2013e). `http://arc.aiaa.org/doi/book/10.2514/4.102301`. AIAA Standard.

[6] Space plug-and-play architecture: Spacewire subnet adaptation (aiaa s-133-9-2013e). `http://arc.aiaa.org/doi/book/10.2514/4.102370`. AIAA Standard.

[7] Virtual network protocol. `http://virtual-network.github.io/vn-sdm/`. Accessed 2013-12-02.