1-1-2012

Scalable Network Approach for the Space Plug-and-Play Architecture

Jacob H. Christensen
David B. Anderson
Mark E. Greenman

Follow this and additional works at: https://digitalcommons.usu.edu/sdl_pubs

Recommended Citation
https://digitalcommons.usu.edu/sdl_pubs/31

This Article is brought to you for free and open access by the Space Dynamics Lab at DigitalCommons@USU. It has been accepted for inclusion in Space Dynamics Lab Publications by an authorized administrator of DigitalCommons@USU. For more information, please contact rebecca.nelson@usu.edu.
Scalable Network Approach for the Space Plug-and-play Architecture

Jacob H. Christensen  
Space Dynamics Laboratory  
1695 North Research Park Way  
North Logan, UT 84341  
435-713-3081  
jacob.christensen@sdld.usu.edu

David B. Anderson  
Space Dynamics Laboratory  
1695 North Research Park Way  
North Logan, UT 84341  
435-232-5760  
david.anderson@sdld.usu.edu

Mark E. Greenman  
Space Dynamics Laboratory  
1695 North Research Park Way  
North Logan, UT 84341  
435-713-3336  
mark.greenman@sdld.usu.edu

Bryan D. Hansen  
Space Dynamics Laboratory  
1695 North Research Park Way  
North Logan, UT 84341  
435-713-3080  
bryan.hansen@sdld.usu.edu

Abstract—The Air Force Research Laboratory (AFRL) is sponsoring an effort to develop Plug-and-Play (PnP) technology for spacecraft systems. The Space PnP Architecture (SPA) supports a method of constructing arbitrarily complex arrangements of components. SPA is a networked data exchange model. This paper presents the SPA network architecture in relation to the standard five layers of the Open System Interconnect (OSI) model. The responsibilities and functionality of each layer are described. The SPA networking provides a unified methodology for self-discovery and self-configuration of heterogeneous PnP networks. The SPA networking approach is shown to be elegant, robust, and scalable.

(An abstract of about 150 words should concisely describe the work being reported, its methodology, principal results and significance.)

TABLE OF CONTENTS

1 INTRODUCTION ................................. 1
2 THE SPA STACK ................................ 2
3 THE PHYSICAL LAYER .......................... 2
4 THE DATA LINK LAYER ......................... 3
5 THE NETWORK LAYER ........................... 5
6 THE TRANSPORT LAYER ....................... 7
7 THE APPLICATION LAYER ...................... 8
8 CONCLUSION / SUMMARY ....................... 9
ACKNOWLEDGMENTS ............................ 10
REFERENCES ................................... 10
BIOGRAPHY .................................... 10

1. INTRODUCTION

The Air Force Research Laboratory (AFRL) is sponsoring an effort to develop Plug-and-Play (PnP) technology for spacecraft systems. The goal is to reduce spacecraft development time and cost by establishing a standards based approach for the development, integration, testing, and operation of spacecraft systems. In a PnP system machine-negotiated interfaces are used to enable components to interoperate. The PnP process of electronic self-discovery and self-configuration eliminates the need to develop specialized hardware and software interfaces for each spacecraft. The Space PnP Architecture (SPA) supports a method of constructing arbitrarily complex arrangements of components. SPA is a networked data exchange model. One of the premises of SPA is that there is no distinction between a hardware device that supports a data interface and a software application that does the same.

A typical spacecraft system involves many different components that vary in bandwidth demand. Sensors that require a very low data rate may reside on a simple two-wire interface such as I²C. Complex sensors, such as an advanced imager, that require a much high data rate may reside on a SpaceWire or optical interface. A spacecraft system is also likely to have a number of components of intermediate performance, with a data rate greater than the simpler sensors, but far simpler than high-performance payloads. A SPA system needs to be able to support multiple types of interconnection networks for hardware and software components that are dramatically different in their addressing schemes and routing capabilities.

Space Dynamics Lab (SDL) has designed an interconnection network agnostic schema which allows a SPA component to communicate with any other SPA component without a priori knowledge of where the component is physically located on the system or what type of interconnection network it uses. The schema has been reviewed and accepted by the SPA standards committee. The schema provides interoperability across existing heterogeneous interconnection networks and a methodology for adding any number of future network technologies without affecting existing SPA components.

The SPA Service Manager (SSM) is an AFRL funded implementation of the AIAA SPA standards including the SPA Networking Standard. The SSM was developed at SDL using SDLs ISO 9001 certified spaceflight software development process. The SSM provides the core services required in order to support PnP. These core services include a mechanism for hardware and software components to publish their data and capabilities with the system. This information is contained in an XML document called an extensible Transducer Electronic Data Sheet (xTEDS). As part of the self-discovery and self-configuration process, a component provides its xTEDS to the SSM. The SSM parses and stores the information contained in the xTEDS. After this process is complete, any component can query the SSM for data or capabilities that it needs. The SSM sends a list of com-

978-1-4577-0557-1/12/$26.00 ©2012 IEEE.  
1 IEEEAC Paper #xxxx, Version x, Updated 27/10/2011.
ponents that can provide the needed data and capabilities. The requesting component can then subscribe to the required data or capability from the providing component of its choice regardless of where it is physically located on the system or what type of interconnection network it uses.

This paper presents the SPA network architecture in relation to the standard five layers of the Open System Interconnect (OSI) model. Section 2 provides an overview and comparison of the SPA network model and the OSI model. It describes the rational and benefits of a layered architecture approach and the primary purpose and function of each of the layers. Sections 3-7 describe how each layer of the model has been implemented in the SPA network design and SSM. Section 8 is a summary and conclusion of the material presented.

2. The SPA Stack
This section will discuss a comparison of the SPA Model to the Open System Interconnect (OSI) model. The base comparison will be performed in this section, with further details on the SPA model implementation provided in further sections.

The OSI model is a layered model of abstraction levels for communications [1]. The OSI model is a broadly recognized and industry standard means of providing an ordered, flexible, and extensible communications system architecture. Each layer of the OSI ISO networking model encapsulates and addresses a different aspect of the communication system requirements.

Use of a layered architecture provides many benefits. In a layered architecture, an instance of a layer provides services to higher level layers and receives services from the layer below. The modularity of the layers promotes ease of understanding the architecture. The design of each layer can be addressed individually, reducing the complexity of the associated communication system. Each layer can be developed independently, can be tested at the layer boundaries, and is easier to maintain. Since the responsibilities of the layer are well categorized, swapping out a layer with an alternate implementation has minimal impact on other layers.

A simplified form of the OSI abstraction layers can be composed into a hierarchy of five abstraction levels, where for SPA comparison the Application layer includes the traditional session and presentation layers as well. Similar levels of abstraction have been applied to the design of the SPA architecture. The layers in the architecture model for SPA are the Physical, Data Link, Network, Transport, and Application levels (see Figure 1).

In the OSI model, the physical layer defines the electrical and physical specifications for devices, and the relationship between a device and a transmission medium. The physical layer provides for conversion between the physical transmission medium and digital data, establishment and termination of connections, and provides for flow control and contention resolution. The SPA Model defines standards for the interconnect between devices in a specific physical subnet type, and includes standards for SpaceWire, and developing standards for I²C and USB.

The OSI data link layer provides the means to transfer data between the network participants, including discovery of physical addressing, and for error correction on errors that may occur in the physical layer. The SPA Model provides similar capabilities utilizing subnet type specific protocols, denoted generically as SPA-X with the X indicating type physical subnet type.

The network layer provides for the transfer of variable length data messages from a source host on one network to a destination host on a different network. The network layer provides routing, and utilizes a logical addressing scheme. Where required, fragmentation and reassembly of messages occurs at this layer. The network layer handles the convergence of messaging traffic from subnet specific transfer to network independent messaging. The SPA Model provides these capabilities through the use of the SPA Logical Messaging protocol.

The transport layer enables the connection and discovery between network end points, and handles reliability for the Application layer. The transport layer also handles message acknowledgement and retransmission. In the SPA model, the discovery and connection is made between clients and services based upon specifications of the end point interfaces in an XML specification known as the xTEDS.

The application layer is the OSI layer which interacts directly with the software applications that implement a communicating component. In the SPA model components can be software applications executing on a general processor resource, or a device on the network. The communication and interoperation is independent of the nature of the resource, and consumers of services are unaware of physical type or physical network location of their producers, enabling seamless plug-and-play.

3. The Physical Layer
A SPA network is composed of several different networking technologies. Each of these are referred to as a SPA Subnet. The SPA standards currently support five different subnets. They are: I²C, USB, SpaceWire, Optical, and local UDP sockets. In order to understand the overall approach of a SPA network it is important to understand the requirements for a SPA subnet and how each networking technology measures

<table>
<thead>
<tr>
<th>OSI Model</th>
<th>SPA Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Applications &amp; Devices</td>
</tr>
<tr>
<td>Data, application-to-application communication</td>
<td>SPA xTEDS Interface</td>
</tr>
<tr>
<td>Transport</td>
<td>SPA Messaging</td>
</tr>
<tr>
<td>End-to-end data discovery</td>
<td>SPA-X Protocols</td>
</tr>
<tr>
<td>Network</td>
<td>SPA-S, SPA-U, etc</td>
</tr>
<tr>
<td>Path determination and logical addressing</td>
<td>Physical</td>
</tr>
<tr>
<td>Data Link</td>
<td>Media, signal and binary transmission</td>
</tr>
<tr>
<td>Physical address discovery</td>
<td>I²C, USB, SpaceWire, Sockets</td>
</tr>
</tbody>
</table>

Figure 1. Comparing SPA to the OSI model
up to those requirements. Each subnet has different capabilities and none of them meet all of the requirements for a SPA subnet and therefore each must be augmented in its own way.

It has often been asked why there are so many different SPA subnets. The answer to that question is best explained by Figure 2. A single SPA subnet is not the solution to all problems. An example of this is a simple temperature sensor. A systems engineer wouldn’t appreciate the power requirements of the SpaceWire interface that would be required to have that single temperature reside on the SpaceWire network. There would also be a huge waste of bandwidth. An I²C bus would be sufficient and have lower power requirements. The inverse of this could be an imager that has many gigabytes of data to transfer. Even though the low power requirements of the I²C may be attractive, the higher bandwidth of an optical interface would be more desirable. A system engineer should choose the SPA subnet that consumes the least amount of power that still meets bandwidth requirements.

There are two primary capabilities that are required of all SPA subnets:

1. Discover currently attached components
2. Send and receive messages asynchronously

Where there is a shortcoming in one of the sub-network, it is overcome in the Data Link Layer. These necessary augmentations are described in Section 4. The rest of this section compares I²C, USB, SpaceWire, and sockets. The optical subnet is not mentioned because it is still under development.

I²C

I²C [2] (Inter-Integrated Circuit) (generically referred to as “two-wire interface”) is a multi-master serial single-ended bus invented by Philips. An I²C physical address is a seven bit address. All devices on the bus receive all communications from all other devices. Each communication is preceded by the seven bit address of the device that is being spoken to. Only that device responds to the communication. I²C does not have the native capability to discover the device on the bus. Due to the master-slave communication paradigm, an I²C bus does not allow asynchronous sending and receiving of messages. This makes the I²C bus the least capable subnet and requires the greatest augmentation of its capabilities.

USB

USB [3] is a two wire bus network that is found on many terrestrial systems including PCs, mobile devices, consumer electronics, and even automobiles. USB was developed by Ajay Bhatt while working for Intel. USB is a master-slave bus where communication is managed by a host controller. USB devices cannot initiate data transfers, but instead only respond to requests given by the host controller. As with I²C, the master-slave nature of USB inhibits devices from asynchronous sending and receiving of messages. However, standard USB host controllers implement a round robin polling loop that allows this to be overcome. This same round robin polling loop allows USB to discover the devices that are currently attached to the bus.

SpaceWire

SpaceWire [4] is a communication network designed for specifically for spacecraft. It is coordinated by the European Space Agency (ESA) in collaboration with international space agencies including NASA, JAXA and RKA. A SpaceWire network consists of nodes that are connected through low-cost, low-latency, full-duplex, point-to-point serial links and packet switching wormhole routers. Because of the full-duplex serial links, SpaceWire components can send and receive message asynchronously. However, SpaceWire uses path based routing and a physical address is determined by the path between two components on the network. This means that the physical address to send from one component to another is totally unique in the network. This makes discovery of the components that are attached to a SpaceWire network very difficult.

Inter-Process Communication using Sockets

The last subnet type discussed in this section is not like the other subnets. Applications running on a processor are treated as though they live in a sub-network. This subnet is the inter-process communication (IPC) network. In SPA the IPC network uses sockets [5] as the transmission media. Sockets were chosen for their portability, pervasiveness, and easy of use. Because there is no physical media to communicate on, the sockets can send and receive message asynchronously quite easily. Each socket is assigned a port to communicate on. The port functions as a sixteen bit address for the software process. This creates a possibility of 65536 possible addresses making it difficult for an application to detect which ports are used by a SPA software process.

4. THE DATA LINK LAYER

The purpose of the data link layer is to provide the means to transfer data between the network participants. To enable this data transfer this layer includes network discovery at a physical addressing level. In a SPA network, facilitation of data transfer and discovery is handled by component known as SPA Subnet Managers.

SPA Subnet Managers

Each physical network type requires a certain degree of management in order to bring that subnet up to the SPA level. This management is handled by software components known as SPA Subnet Managers. A SPA subnet manager has several key responsibilities in a SPA network. It is responsible for discovering the components within its subnet, assigning each
component a unique logical address, routing messages in and out of its subnet, and monitoring the health and state of those discovered components. Each SPA Subnet being managed by a SPA Subnet Manager is associated with a different physical transport medium: SPA-Local uses IPC sockets, SPA-E uses Ethernet, SPA-U uses USB, SPA-1 uses I2C, and SPA-S uses SpaceWire. This set is expandable as different physical transport networks are brought into SPA. It is also important to note that because a SPA Subnet Manager is a software process, it participates on both the IPC subnet of the processor on which it executes, as well as on the physical transport subnet which it is managing.

The first task that a SPA Subnet Manager must complete is to alert the subnet manager of the local IPC subnet of its presence and its needs of a unique address block for its subnet. This address block will allow the SPA Subnet Manager to assign unique addresses from within that block to the components it discovers and manages on its subnet.

The second task a SPA Subnet Manager must accomplish is discovering the components that live on the managed subnet. The discovery protocols differ for each subnet type due to the different capabilities each provides. An example of these differences can be seen when comparing IPC and I2C. A software component utilizing IPC can proactively alert the SPA-Local Subnet Manager of its presence on the subnet upon startup. In contrast, a device on an I2C bus is a slave and cannot initiate communication with the SPA-1 Subnet Manager. Therefore, a SPA-1 device must wait to be discovered by the SPA-1 Subnet Manager. This means that a SPA Subnet Manager is responsible for bridging the gap between its subnets inherent capability and the requirements for a SPA Subnet. Once a SPA Subnet Manager has discovered a device it can assign that device a SPA logical address from within its assigned address block.

**Discovery**

Component discovery is the process in which a component is found and assigned a logical address on the network. Each subnet manager is required to discover the components on their subnet. This process is different for each subnet due to the inherent differences in the capabilities and topologies of a subnet medium. The discovery process for each subnet is outlined here.

**SPA-Local**—The SPA-Local subnet consists of applications running on a single processing node in the network. A distinct SPA-Local subnet exists for each processing node in the network. These applications consist of user software, SPA Subnet Managers, and any other software processing wishing to participate on the SPA network. A SPA-Local subnet utilizes sockets as an IPC mechanism to allow these applications running on the same processor to communicate with each other. Because usage of a socket is not inherently discoverable, a protocol is needed to discover applications and configure this subnet. The SPA-Local protocol consists of a set of messages specific to the SPA-Local subnet. These messages contain physical addressing information (ports) and never leave the SPA-Local subnet. These SPA-Local messages are prefixed with Local to differentiate between these subnet-specific messages and the network-wide SPA messaging prefixed by Spa.

The SPA-Local Subnet Manager listens on a well-known port so that applications can alert the SPA-Local Manager of their presence. When a SPA-L component comes online, it does this by periodically sending a LocalHello message to the SPA-L Subnet Managers well-known port. This LocalHello contains the port on which the application is listening to establish two-way communication. The SPA-Local Subnet Manager will respond with a LocalAck so that the application knows to cease sending LocalHello messages. The SPA-Local Subnet Manager can then assign the SPA-L component a logical address from within its allocated address block using the SPA-AssignAddress message. One important thing to note about the SPA-Local subnet is that all SPA Subnet Managers are SPA-Local components. This means that SPA Subnet Manager also report their existence to the SPA-Local Subnet Manager on the processor on which they reside. A field in the LocalHello message allows the sender to specify their component type. If this field is set to Subnet Manager, then the SPA-Local Subnet Manager does not assign an address from within its address block, but instead requests a new address block in behalf of the newly discovered SPA Subnet Manager. This process will be discussed in greater detail when the Central Addressing Service is discussed.

**SPA-I**—A SPA-1 subnet consists of devices connected to an I2C bus attached to a processing node in the SPA network. As previously discussed, I2C components are addressed with a 7-bit address which must be unique on the bus. The problem is that there is no bus arbitration on I2C for slaves, if two slaves attempt to utilize the same address, their data will become mixed and corrupted as both attempt to transmit on the bus at the same time. Generally avoiding duplicated slave addresses is accomplished by either modifying the firmware configuration of a device or selectively applying power to certain pins. This type of manual bus configuration is not acceptable when needing to attach multiple devices produced by different vendors to the same bus. To solve this we have designed an Address Resolution Protocol (ARP) which functions on standard I2C for SPA-1 which allows the set of SPA-1 components to self-organize their physical address space. Figure 3 shows a sequence diagram of the SPA-1 ARP for three components.

**SPA-1 ARP Algorithm:**

1. All SPA-1 components start out with their address initially set to 0x11.
2. Every SPA-1 component switches to master mode and attempts to send a message which contains a universally unique identifier (UUID) to the address they have selected (0x11 initially).
3. Because of this UUID, there are bits guaranteed to be different between each of these messages and therefore standard I2C master bus arbitration ensures that only one component may successfully transmit this message.
4. Every component that receives this message must acknowledge receipt per the I2C specification.
5. The component that was able to successfully send the message checks to see if the message was acknowledged or not.
   (a) If no acknowledge, the component keeps the address it has set and starts normal operation.
   (b) If an acknowledge bit is received, the component increments its address by 1 and starts the algorithm again

After several iterations all components will have unique addresses beginning at 0x11 and completely filling the address space above that. This efficient use of the address space ensures that a maximum number of components may exist on the network. Race conditions are taken into account by having each component switch their address to the address they intend to take before sending the message which checks
A flooding algorithm is required to perform topology discovery on a SPA-S subnet made up of industry standard SpaceWire components. The basic idea behind the algorithm is to utilize the properties of the wormhole path-based routing scheme to discover routers and the outputs ports to which components are connected. After finding a router, communication is then attempted with all possible router ports to discover other routers or components connected to those routers. This can be performed from any perspective within the SPA-S subnet, but would generally only be performed by processing nodes.

**SPA-S Discovery Algorithm:**

1. Send a batch of SpwRouterProbe messages containing the UUID of the sender to all possible output ports of connected router. When sending SpwRouterProbe messages they are always sent along a path that will result in reflecting the sent message back to the sender. This path information must also be stored in the message payload as it will be stripped off while being routed through the network.

2. Receive SpwRouterProbe messages reflected back from the subnet. The contained UUID must be checked to ensure it matches as others on the subnet may be performing this same process simultaneously. Receipt of a reflected-back SpwRouterProbe indicates discovery of a SpaceWire router.

3. Send a batch of SpwEndpointPing messages to all possible router output ports of the discovered router with the exception of port on which the router was discovered. SpwEndpointPing message payloads must contain both the forward and return SpaceWire paths.

4. Components (including pieces of the SPA-S Subnet Manager) respond to received SpwEndpointPing by sending SpwEndpointPingReply messages to the included return path.

5. A component is discovered on receipt of SpwEndpointPingReply messages.

6. Send another batch of SpwRouterProbe messages to each output port of the discovered router, each formatted to be reflected back if a router exists at that location.

7. Continue the process above until no more SpwRouterProbe messages are reflected back.

**5. The Network Layer**

The first step in combining each SPA subnet into a single SPA network is to define a transport agnostic logical address. Because the logical address is transport agnostic, none of the subnets have to know about the physical addressing of any of the other subnets. This allows the logical address to stay the same no matter where or how it traverses the SPA network. In order to prevent address collision a single entity known as the Central Address Service (CAS) works in conjunction with the SPA subnet managers to ensure that unique address assignments are made. Second, SPA subnets have to have a way to communicate with each other in order to route messages through the network. Because each SPA subnet manager is a software process, the SPA-Local (IPC) subnet is used as the common language for all SPA subnet managers. Finally, even though a transport agnostic logical address has been assigned, the physical addresses are still need to perform actual communication on the network. A routing table is defined and a routing protocol to use it is described.
SPA Logical Address

In selecting a transport agnostic logical addressing scheme the driving goals were to keep it simple and have a low overhead. If an address is too large then more time is spent transmitting addresses instead of data. For these reasons a SPA logical address was chosen to be 4 bytes in size. The 4 bytes are divided in half with the upper two bytes being the SPA subnet ID and the lower two bytes being the component ID. This approach lends itself to representing SPA logical addresses in an ordered pair notation (i.e. (3,2)). It allows assignment of addresses to happen in two phases.

Central Address Service and Address Assignment

During discovery every component on the network is given a SPA logical address. This process takes place in two phases. When a SPA subnet manager is discovered, it is assigned an address block by the Central Address Service (CAS). The CAS ensures that each address block assignment is unique. An address block is a normal SPA logical address, except that its component ID is 0. This means that every SPA subnet manager has the first address in their address block (i.e. (1,0), (2,0), (3,0), ... etc.). Once a SPA subnet manager has its address block, it can assign addresses within its own subnet without having to consult the CAS. Each address assignment is sequentially assigned. The SPA subnet manager who has assigned the address block (4,0) would assign the address (4,1) to the first discovered component in its subnet. The subsequent address assignments would be (4,2), (4,3), (4,4), ... etc. Figure 5 shows an example network with SPA logical address assignments and a routing path. A SPA network can contain up to 65536 different subnets, each containing 65536 individual components.

The SPA-Local Interconnect

SPA-Local is different than the other SPA subnets because it functions as an interconnect between all other SPA subnets. Each SPA subnet manager speaks two protocols: SPA-Local and their own SPA-X protocol. This allows a message originating from the SPA-1 subnet to get to the SPA-S subnet by traversing the SPA-L subnet.

Routing Tables and Routing

The SPA subnet managers also act as routing entities in a SPA network. Each subnet manager is responsible for maintaining a routing table. The routing table contains the information necessary to translate a logical address into the appropriate physical address to move a message on to its final destination. A routing table contains two different sets of entries. The first set contains entries for all of the other SPA subnet managers in the SPA network. The second set contains entries for all of the components attached to the manager's subnet. When a message is received by a SPA subnet manager it follows a very simple algorithm:

```
if dest[subnetId] == mySubnetId
    send to component on my subnet
else
    send to manager of dest[subnetId]
endif
```

In order for the routing table to contain all the needed entries, when a SPA subnet manager finds another SPA subnet manager, it sends a message to all other known SPA subnet managers telling them "if you want to talk to this new subnet manager, talk to me". This behavior creates a path from any subnet manager to any other subnet manager.

<table>
<thead>
<tr>
<th>Logical Address</th>
<th>UUID</th>
<th>Address Type</th>
<th>Physical Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>3,0</td>
<td>0x324</td>
<td>Self</td>
<td>Self</td>
</tr>
<tr>
<td>3,1</td>
<td>0x983</td>
<td>SpW</td>
<td>3</td>
</tr>
<tr>
<td>3,2</td>
<td>0x873</td>
<td>SpW</td>
<td>4</td>
</tr>
<tr>
<td>3,3</td>
<td>0x907</td>
<td>SpW</td>
<td>5</td>
</tr>
<tr>
<td>0,1</td>
<td>0x908</td>
<td>IPC</td>
<td>1982</td>
</tr>
<tr>
<td>1,0</td>
<td>0x948</td>
<td>IPC</td>
<td>1983</td>
</tr>
<tr>
<td>2,0</td>
<td>0x234</td>
<td>IPC</td>
<td>1984</td>
</tr>
<tr>
<td>4,0</td>
<td>0x842</td>
<td>Ref</td>
<td>3,3</td>
</tr>
<tr>
<td>5,0</td>
<td>0x973</td>
<td>Ref</td>
<td>3,3</td>
</tr>
</tbody>
</table>

Figure 4. An example routing table

The routing table contains four items in each entry:

- Logical address
- UUID
- Address type
- Physical address

In Figure 4 an example routing table is shown. The first four entries are the individual components on the manager's subnet. It is apparent that this routing table belong to a SPA-S subnet manager. It can also be seen that the SpaceWire network only has one router. This is evident by the fact that the SpaceWire routes stored in the routing table only have a single byte for the SpaceWire path. The following routes are for the CAS and other SPA subnet managers. From the routes for the addresses (4,0) and (5,0) it is apparent that these subnet managers are not on the same processing node as this SPA-S subnet manager. In fact, if the reader looks closely it can be seen that this is the routing table for the SM-S with the address (3,0) from Figure 5.

As an example of how routing works in a SPA network, the SPA-1 component with address (2,2) is going to send a message to the SPA-U component with address (5,3). First, the SPA-1 component sets the message destination to (5,3) and the message source to (2,2). However, since it cannot send directly to the SPA-1 subnet manager it has to wait for its turn in the round robin polling loop. Once it has been contacted it sends the message to the SPA-1 subnet manager. The SPA-1 subnet manager sees the destination subnetId is 5 and looks up in its routing table the physical address to send a message destined to the SPA subnet manager with the subnet ID of 5. It sends the message on to the SPA-S subnet manager with the address (3,0). The SPA-S subnet manager looks up the next address in the path and sends the message through the SpaceWire network to the secondary SPA-S subnet manager with address (3,3). The secondary SPA-S subnet manager with address (3,3) looks up the next address and send the message to the SPA-U subnet manager over the SPA-Local network. The SPA-U subnet manager sees that the message is destined for a component on its subnet and sends the message to its final destination, the component with address (5,1).
6. THE TRANSPORT LAYER

The purpose of the transport layer is to enable the connection and discovery between network components. In a SPA network this involves resolving a component's data dependencies with those items currently available in the network. This is accomplished through registering descriptive data sheets embedded within each component with a centralized repository and then issuing queries to that repository to locate the desired dependency.

xTEDS

Each component on a SPA network describes itself and its capabilities through an embedded document known as an eXtended Transducer Electronic DataSheet (xTEDS). xTEDS extend the concepts of the IEEE 1451 TEDS standard by not only including identification, calibration, and manufacturer information, but also describing the components data inputs and outputs. All of this information is stored in XML format. Using XML allows the xTEDS to be human readable, but also easily machine-parseable. Any xTEDS document can also be verified for correctness by validating it against the xTEDS XML schema. The purpose of these xTEDS in a SPA network is to allow components to discover their data needs or dependencies dynamically. This is achieved by utilizing a common set of terms and interface definitions defined in what is known as the SPA Ontology. Figure 6 contains a portion of an xTEDS.

Lookup Service

The Lookup Service is a critical component on a SPA network which acts as a repository and query engine for all xTEDS on the network. After a component has successfully been discovered and assigned a logical address, it must then register with the Lookup Service. This process involves having the SPA Subnet Manager who discovered the component inform the Lookup Service of that discovery. The Lookup Service will then decide if that component's xTEDS should be requested. This allows the Lookup Service to not request an xTEDS it already has in its repository, saving network bandwidth and processing time. An xTEDS might already be present if multiple components present the same xTEDS to the system or if the xTEDS were cached from a previous run.

Query

Any component in the system may issue queries to the Lookup Service to find data or components to satisfy their dependencies. These queries are issued using the common terms defined in the SPA Ontology and providing a desired value to associate with those terms. These terms are represented as XML attributes in the xTEDS. If the values associated with the attributes are numeric, the query may use standard arithmetic operations to specify desired ranges for certain quantities. An example would be precision \( \geq 2 \) (Figure 7). This query would return any value in the system with a precision greater than or equal to two. Lookup Service is responsible for responding to queries with a stream of matches available on the network. When multiple specifications are present in a query, they combine conjunctively, so the result is the intersection of the results that would be returned if each specification were present individually. A very specific query will return fewer results, whereas more generic queries will return more results. To eliminate race conditions among registrations, queries can also be made for future registrations. This allows the component who issued the query to be notified of any future additions to the network that may better meet its needs.

Each match contains the logical address of the component, the byte-level format of the provided message, and the section of xTEDS which defines the message. This information allows the issuer of the query to make a selection of which source to utilize. At this point the issuer of the query has sufficient information to issue the selected commands to the component or request a subscription to the provided data.

**Figure 5.** Routing a message through a SPA network

**Figure 7.** A very simple query

<SpaQuery msgType="Notification"><Variable><Attribute name="kind" operand="eq" value="temperature"/> <Attribute name="precision" operand="gte" value="2"/></Variable></SpaQuery>
Publish/Subscribe

Data transfer within a SPA network is done primarily using the publish/subscribe paradigm. After issuing a query and selecting a data provider, components may issue subscription requests to the provider of that message. Along with requesting a subscription to a piece of data, the requester may also request a specific lease period for which the subscription will remain valid. Lease periods provide a degree of fault tolerance as the subscriber can detect a failed data provider and select a different data source to subscribe to. The subscriber should renew this subscription before the lease period expires to ensure continued data flow. If a component produces the data at a rate higher than the subscriber desires, the subscriber may also specify a delivery rate divisor to indicate that it should only publish one of every n messages to the subscriber. Data producers or publishers have the ability to reject any requested subscription to their data based on priority, resources, or data availability.

7. THE APPLICATION LAYER

The SPA architecture layers described previously all culminate in support for SPA components to interoperate in a plug-and-play manner at the SPA Application layer. This section will describe the life cycle of a SPA component.

In the SPA model, a SPA component is an endpoint whose interface conforms to the SPA standards, and which does not connect to another SPA object via a different port. SPA components can be software applications executing on a general processor resource, or a device (hardware) on the network. These may include applications for Guidance, Navigation and Control, power management, payload management and operation, system health and status, etc. The communication and interoperation of the SPA components is independent of the nature of the resource. Consumers of services are unaware of physical type or physical network location of their producers.

The core concept of SPA is that components register their capabilities with a Lookup Service when they are added to the system. Once this information is captured, any component with a data need may query the Lookup Service for available sources and receive matches to that query. Subsequently, that component may contact any or all matching components directly to subscribe to data that it provides or utilize its data services.

A SPA component goes through a five phase life cycle. The phases in the life cycle are:

1. Component Discovery
2. Component Registration
3. Data Source Query
4. Data Subscription
5. Normal Operations

After establishment of subscriptions, the component is prepared to enter normal operations. Services that are required

---

Figure 6. Example portion of simple xTEDS

Figure 8. SPA Component life cycle
by a SPA component have been matched to providers, and requests and subscriptions received from other SPA components can now be serviced. As a dynamic system, providers may become unavailable. The subscription protocols and Lookup Service design provide for notification of the drop-out to any consumers of the providers services. Similarly, a SPA component may become newly or again available. Consumer components which have queried previously for services that the newly registering component provides will be notified, and can respond if the new source is deemed better or still required. Component discovery is the process in which a component is found and assigned a logical address on the network. This has previously been discussed as part of the Data Link layer. Each subnet discover the components on their subnet using the process.(probing, listening, etc) appropriate for the subnet topology and medium. Upon assignment of a logical address, a component is ready to participate in a SPA network, and transitions to the registration phase.

In the Component Registration phase, the component informs the Lookup Service of its available interfaces, including commands available to configure and control the component, data produced by the component which is available to the system, and services which include a request/response from the component. This interface specification is provided to the Lookup Service, and any other SPA component in the system which requests it, as an xTEDS. Upon completion of registration with the Lookup Service, a component and its services are now known, and available for use by other components which search for compatible services.

After registration, the component enters a query phase, where it will search for services in the system that it requires. The SPA architecture provides logical messaging protocols which support query for services and response messages to the query. The component will issue a query to the Lookup Service. The query does not search for components by name or physical location, but rather through the type of service required, with some available associated qualifiers. The query mechanism uses a XML schema very similar to that used for the xTEDS. The Lookup Service will return a response to the query including a list of components by logical address that provide the requested data services. Depending on the query specification, the responses may be ranked using supplied ranking criteria. In addition, the query may remain in effect in the Lookup Service so that any matches to the query request that become available after the initial query response will also be forwarded to the querying component. Some components, may be only a producer of system information. For example a controller of set of temperature monitors may only report information to subscribers, and not require issuance of any queries to locate other services. Simple hardware devices would typically fall into this category of SPA component. For SPA components which are consumers, once query responses have been received, they are ready to select and subscribe to services.

Having received a list of matches to queried for services, a SPA component is now ready to select and subscribe to data and service producers. A ranking of matches may be included in the query response, but the SPA component is free to utilize whichever selection mechanism is most appropriate for it. The SPA component can request the use of the service of the providing component either in an ad hoc manner, or it may use SPA protocols to establish a subscription to the service or data for a time period. Similarly, a SPA component which provides data services may receive requests for those services, and subscriptions. The SPA component will acknowledge to the requester whether the request will be honored, and prepare to support servicing subscriptions.

After establishment of subscriptions, the component is prepared to enter normal operations. Services that are required by a SPA component have been matched to providers, and requests and subscriptions received from other SPA components can now be serviced. As a dynamic system, providers may become unavailable. The subscription protocols and Lookup Service design provide for notification of the drop-out to any consumers of the providers services. Similarly, a SPA component may become newly or again available. Consumer components which have queried previously for services that the newly registering component provides will be notified, and can respond if the new source is deemed better or still required.

8. CONCLUSION / SUMMARY

This paper has described the SPA network architecture in relation to the standard five layers of the OSI model. It described the benefits of a layered architecture approach and the primary purpose and function of each of the layers. It described how each layer of the model was implemented in the SPA network design and SSM.

The SPA networking approach described in this paper provides a unified methodology for self-discovery and self-configuration of heterogeneous PnP networks. It provides a method for components to publish and subscribe to resources and interoperate regardless of their physical location on the system or the type of interconnection network they use. It allows spacecraft component providers to design and develop SPA compliant components without any a priori knowledge of how or where they will be utilized in the system thus enabling true PnP functionality. It also provides a well defined methodology for adding new and future network technologies without affecting existing SPA components. Support for a new or future interconnection network can be added by simply including the appropriate subnet manager software module at the data link layer. Likewise, removing support for an interconnection network that is not required simply requires removing the appropriate subnet manager software module. The current SSM implementation supports SPA Local (SPAL) and SpaceWire (SPA-S) subnets. Efforts are currently underway to incorporate SPA-1 (I²C), SPA-U (USB), SPA-O (optical), and SPA-C (CAN) subnets as well.

The SPA networking approach has been implemented as part of the core SPA services in the SPA Services Manager (SSM) developed by SDL. The engineering feasibility of this approach has been demonstrated on multiple laboratory prototype implementations. These include a SPA SSM development testbed located at SDL, A PnPsat-2 satellite structure at the AFRL Responsive Space testbed (RST), and an APT contractor satellite structure. Both the PnPsat-2 and the APT contractor satellite structures represent full-scale, realistic prototype platforms on which the SPA networking approach has been integrated and tested. The APT contractor prototype structure was specifically designed to characterize and evaluate the performance of a desired PnP configuration in terms of performance, including physical, logical, and data. The APT contractor structure was based on spaceflight heritage and comparisons between the tested environment and operational environment are analytically understood. Analysis and test measurements quantifying throughput, scalability, and reliability were conducted.
The SPA networking approach has been shown to be elegant, robust, and scalable.

ACKNOWLEDGMENTS
The authors would like to thank the Air Force Research Laboratory for the funding that has supported this work. They would also like to thank the Advanced Plug-and-Play Technology (APT) working group for allowing them to participate and listening to so many of their ideas.

REFERENCES


BIOGRAPHY

Jacob H. Christensen received his B.S. degree in Computer Science from Brigham Young University — Idaho in 2008 and is currently working on his Ph.D. in Computer Science at Utah State University in Logan, Utah. He is also a full-time employee at the Space Dynamics Laboratory. His research interests include self-configuring networks and automatic software generation.

David B. Anderson is a senior Software Engineer for the Space Dynamics Laboratory. He received his B.A. degree in Computer Science from Weber State University in 1987 and his B.A. in Mathematics from Weber State University in 1994. He has designed and implemented flight and ground segment software solutions for numerous aerospace systems since 1995. His research interests are in embedded real-time applications, device drivers, instrumentation, integration, ground support equipment, and verification and validation software.

Mark E. Greenman is a senior Software Engineer for the Space Dynamics Laboratory. He received his B.S. degree in Physics from Weber State University in 1986, and has designed and implemented flight and ground segment software solutions for numerous aerospace systems since 1991. His research interests include applying agile software methods to embedded systems and small satellite software engineering.

Bryan D. Hansen is a Software Engineer for the Space Dynamics Laboratory. He received his B.S. degree in Computer Science from Utah State University in 2009 and is currently pursuing a M.S. in Computer Science also from Utah State University. His research interests include software reuse and abstraction in embedded software systems along and self-configuring networks.