26th Annual INCOSE International Symposium (IS 2016)
Edinburgh, Scotland, UK, July 18-21, 2016

A Representative Application of a Layered Interface Modeling Pattern

Peter M. Shames
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91009
+1 818-354-5740
Peter.M.Shames@jpl.nasa.gov

Marc A. Sarrel
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91009
+1 818-393-7786
Marc.A.Sarrel@jpl.nasa.gov

Annotations - 29Feb24
Added notes for each figure naming the appropriate RASDS++ Viewpoint (and View, where appropriate).
Uses RASDS: Connectivity (component & connector), Communications (protocol stack & behavior), and Information viewpoints, along with various SysML diagram types (IBD, State Chart, Activity).
Reference CCSDS 311.0-M-1 (in revision)

Sanford Friedenthal
SAF Consulting
Affiliate, Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91009
Sanford.Friedenthal@gmail.com

Abstract. Model-based systems engineering (MBSE) is intended to improve how systems engineering is performed compared with a more traditional document-based approach by effectively using models to analyze, specify, design, and verify systems. The OMG Systems Modeling Language (OMG SysML™) enables the practice of MBSE by providing a robust and expressive language for representing systems.

Several MBSE methods are available [3], and have continued to mature over the last several years which include model-based practices for requirements flow-down, architecture-design, trade-off analysis, verification planning, and others. One of the critical systems engineering practices is interface modeling. This paper describes a layered interface pattern for modeling data and communications interfaces using SysML. The pattern spans logical to physical interface definition, and includes software and electrical interfaces.

Each layer in a stack describes a portion of the interface functionality. The concept of a layered interface is borrowed from computer networking [8] [10]. The layered interface pattern described in this paper enables the specification and design of connections and behavior between interfacing systems at a given layer, and between the adjacent layers of a single system. This pattern may also be applied recursively. That is, communication within a single layer may itself be realized by a multi-layer stack. The level of detail of the model to describe a layered interface should be adapted to the need, and can vary from highly abstract logical flows across a system to highly detailed protocol specifications and message structures.

This paper builds on work that was documented in a previous paper entitled "*A modeling pattern for layered system interfaces*" [5]. Aspects of this pattern have been demonstrated in various project applications including Exploration Flight Test 1 (EFT-1), Space Communication and Navigation (SCaN) Trade Studies, and the SCaN Network Integration Project (SNIP).

Introduction

This paper is organized into six sections. The Introduction, section 1, describes some of the challenges associated with interface specification and design. This section then introduces some basic definitions and concepts that the layered interface pattern uses and provides a simple example. It concludes with a brief overview of SysML.

The System Example, section 2, uses a spacecraft example to describe a logical data flow through a system, and how data interface requirements can be specified and allocated to various parts of a system.

The Interface Realization, section 3, describes how a particular interface from the spacecraft example can be realized by a layered protocol stack that converts the data contained in communication packets to electrical signals on a physical connector, which are exchanged through a physical medium. The modeling pattern defines each vertical layer of the stack, the data structure that is transformed from one vertical layer to the next, the behavior specification for an example layer, and the interaction between peer layers at each side of the interface. The sections ends with an example of how to show compliance with a standard that defines a protocol.

Related Work, section 4, discusses some related work. Applications, section 5, then discusses how this pattern can be applied more generally to other types of interfaces, and finishes with the Summary, section 6.

This pattern is capable of representing multi-layer interfaces at varying levels of detail. The level of detail should be adapted to the need. Sometimes, only abstract end to end flows may be appropriate to describe the interfaces. In other cases, the level of detail may include some combination of detailed message structure, pin to pin connection, and protocol behavior specification. This is often dependent on the phase of development, and whether the interface is using well understood interface standards or new or modified interfaces are being developed.

The Interface Challenge

Well-defined interfaces are essential to specify how a system can interact with the external world, and how the system elements can interact to achieve the objectives of the whole system. Specifying and designing interfaces is a critical and challenging aspect of systems engineering due to the number of interfaces, diversity of interfaces, and the inherent complexity of individual interfaces. For example, electrical harnesses can contain thousands of wires and connectors and perhaps millions of messages. Many system failures have been attributed to inadequate interface specification and design. [20]

A typical system, subsystem, and component interface is often specified in an interface requirements document (IRD) or similar document. An example of a partial table of contents from a NASA interface control document for a C-9B aircraft in support of the Reduced Gravity Program [4] is included in Figure 1.

This interface control document includes many different types of interfaces including electrical power, high-pressure gas, cabin environment, display interfaces, physical dimensions, and others. These interfaces are realized by many different engineering disciplines using many different technologies. A systems engineer must be able to specify, analyze, and verify interfaces that span the various engineering disciplines and technologies to ensure the elements of the system can work together to achieve the system requirements.

Interface specification and design is not only complex because of the number of interfaces and the many different kinds of interfaces, but any given interface can be complex in its own right. For example, the interface for a “simple” USB device is defined by the Universal Serial Bus revision 2 (USB 2) specification [6] that is 650 pages long, and includes specification of the data flow model, mechanical and electrical interface, and protocol layer.

A model-based approach provides an opportunity to address the challenges of specifying, analyzing, designing, and verifying interfaces over a more traditional document based approach by enhancing consistency, precision, traceability, conformance to standards, reuse, and managing the inherent complexity of interfaces.

1.0 INTRODUCTION
1.1 Purpose
1.2 Scope
2.0 FACILITIES PROVIDED
2.1 Aircraft
2.1.1 Cabin Environment
2.1.2 Cabin Dimensions
2.1.3 Cabin Provisions
2.1.4 Electrical Power and Interface
2.1.5 Aircraft Lighting
2.1.6 High Pressure Gas System
2.1.7 Overboard Vent System
2.1.8 Aircraft G-Load Display
2.1.9 Accelerometer Signal
2.1.10 On-Board Tools
2.1.11 On-Board Storage Containers

Figure 1. Partial Table of Contents for a NASA Interface Control Document [4]

Definitions and Concepts

The following definition of interface is from the Glossary of Terms in the Guide to the Systems Engineering Body of Knowledge [7].

Interface:

1. A shared boundary between two functional units, defined by various characteristics pertaining to the functions, physical signal exchanges, and other characteristics. (ISO/IEC 1993)
2. A hardware or software component that connects two or more other components for the purpose of passing information from one to the other. (ISO/IEC 1993)
3. To connect two or more components for the purpose of passing information from one to the other. (ISO/IEC/IEEE 2009)

The first definition is the most general of the three definitions above because it does not limit interface to the exchange of information. An interface provides the means for systems and system elements to interact, which may include the exchange of information, material, forces, and energy. To specify an interface, one must specify the connection points on the components (i.e. ports) on either side of the interface, the items that are exchanged, the constraints and/or rules that govern the exchange, and the medium for the exchange (i.e., link). An interface definition sometimes refers to one side of an exchange, but more generally refers to both sides of the exchange and the exchange medium. The system, subsystem or other system element (e.g. component) behaviors realize the interface to achieve the exchange. These interface concepts are illustrated in Figure 2.

This paper also uses the term protocol, where a protocol at layer (N) can be defined as a set of rules and formats (semantic and syntactic) which determines the communication behavior of (N)-entities in the performance of (N)-functions. [10] The protocol would be most strongly reflected in the Behavior and the Constraint.

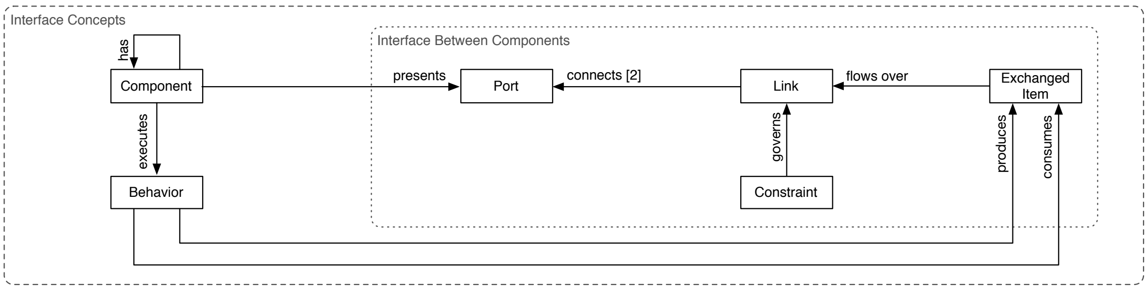


Figure 2. Interface Concepts

Simple Example of a Layered Interface

Figure 3, shows a set of components: a USB digital audio interface between an Audio Player component such as a CD player, an Amplifier component that amplifies the audio electrical signals and converts the digital signal to an analog signal, and a Speaker that converts the analog audio electrical signals to acoustic waves (sound). Each of these interfaces is shown with their protocol stack.

The ability to describe interfaces at different levels of abstraction is essential to address interface complexity. An interface layer[1] is an abstraction approach to help deal with this complexity where each interface layer provides specific functionality associated with the interface. A Protocol Stack is a set of layers that transforms items to enable their exchange, such as for purposes of communication.

RASDS Communication (protocol stack) Viewpoint, showing Correspondences to Connectivity View objects (components).

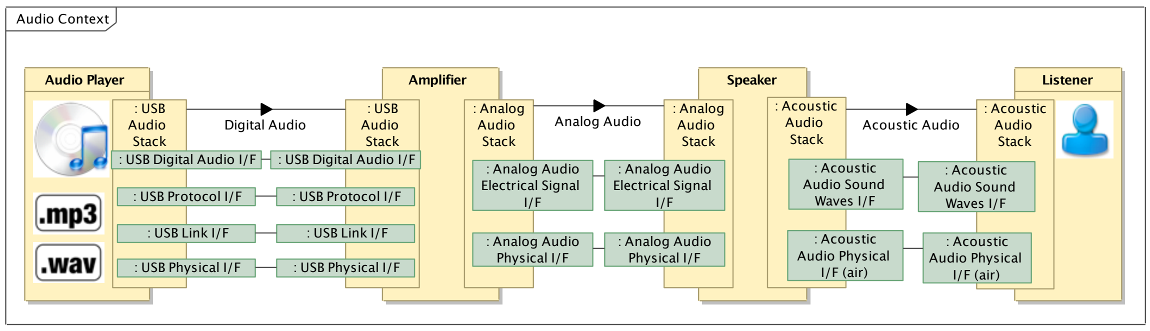


Figure 3: Example of Audio System interfaces

A fundamental principle of an interface layer is that the layer below is independent of the layer above. Consider the connection between the Audio Player and the Amplifier for the USB Digital Audio I/F layer. That layer encodes the digital audio with a certain number of bits per sample, samples per second, number of channels (mono or stereo), etc. The USB Protocol I/F layer below does not know or care about those details. The USB Digital Audio I/F layer will impose quality-of-service constraints on the USB Protocol I/F layer, in

[1]NOTE: The terminology that is used for interface layer is adopted from the ISO/IEC Basic Reference Model (ISO BRM), reference [16] as included below:
5.2.1.2 (N)-layer: A subdivision of the OSI architecture, constituted by subsystems of the same rank (N).
5.2.1.9 (N)-protocol: **A** set of rules and formats (semantic and syntactic) which determines the communication behavior of (N)-entities in[1] the performance of (N)-functions.

particular constraints related to isochrony, minimum throughput constraints, and other constraints. But, the content and format of the audio data is opaque to the USB Protocol I/F layer.

Each of the four layers of the Audio Player and the Amplifier USB Audio Stack performs an orthogonal part of the functions needed to transfer the Digital Audio data. For example, the USB Digital Audio I/F layer is responsible for encoding the audio. The USB Protocol I/F layer is responsible for complete and isochronous delivery of the data. The USB Protocol I/F layer may simultaneously handle types of data other than Digital Audio from different higher level protocols. These top two layers do not allow the presence of intermediate systems between the Audio Player and the Amplifier. The lower two layers, however, would allow such systems. The USB Link I/F layer transmits data between one USB device and another, but with no regard for retransmission or completeness. It may allow intermediate devices like USB hubs. But, those hubs are transparent to the upper two layers. The USB Physical I/F layer is concerned simply with the cable. It may allow intermediate systems like USB extension cords that are not visible to the upper layers.

In this example the Audio Player can handle stored audio data in several formats, for example audio CDs, .mp3 files and .wav files. The format in which the audio data is stored, however, is not relevant to the format in which it is transmitted. In each case, the Audio Player transforms the audio from the original storage format into the USB Digital Audio format for transmission.

An interface view is another abstraction approach to deal with interface complexity. For example, in the digital audio interface above, noise immunity and component proximity may be important concerns for the design of this interface, which can drive specific design choices. The separation between the Audio Player and the Amplifier that is tens of meters instead of 1-2 meters may require a different interface design using a digital audio TOS link fiber optic cable instead of USB. The design decisions must be considered from the perspective of different stakeholder viewpoints that may include different engineering disciplines such as electrical, mechanical, and software perspectives. An interface view presents the interface information that addresses a particular stakeholder viewpoint.

In this paper, we present a modeling pattern that applies to data and communication interfaces that includes logical interfaces, software interfaces, signal interfaces, and physical connections. Although this is applied to communication interfaces in this paper, the pattern can be applied to other kinds of interfaces as well.

Describing the layered interface modeling pattern using SysML is the subject of this paper. However, the pattern reflects layered interface concepts that are well established and have been applied many times to data and communications interfaces. These include the ISO Basic Reference Model [10] and the Reference Architecture for Space Data Systems (RASDS) [14], and the entire Internet protocol suite. RASDS is particularly relevant for the primary spacecraft example used in this paper.

RASDS defines five architectural viewpoints. Three of the RASDS viewpoints are related to the layered interface concept. The Connectivity view shows the lowest physical layer physical view of the nodes and links in the system over which data is routed. The Communications view shows in detail the protocol stacks in detail that perform data communication over the physical medium. The Information view shows the details of how data is packaged. Multiple system views modeled from these viewpoints provide the overall description of the system components, their communications, and behaviors.

SysML Overview

SysML is a general purpose modeling language for modeling systems and their environment that may include hardware, software, data, people, facilities, and natural objects. The language is often characterized in terms of four pillars as indicated in Figure 4 that represent the system requirements, structure, behavior, and parametrics.

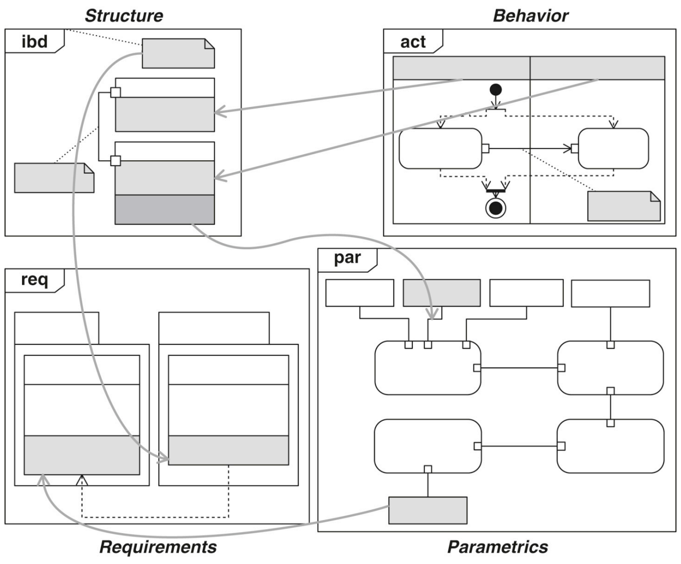


Figure 4. Four Pillars of SysML
From 'A Practical Guide to SysML, 3rd Edition' (Figure 2.1)
Copyright © 2015, 2012, 2009 Elsevier Inc. All rights reserved.

The four pillars of the language include the capability to represent:

- **Structure:** Structural composition, interconnection, and classification
- **Behavior:** Function-based, message-based, and state-based behavior
- **Parametrics:** Constraints on the physical and performance properties
- **Requirements:** Requirements and relationship to other requirements, design, analysis, and test cases

SysML includes the nine kinds of diagrams. The diagrams provide pre-defined ways to present the design of a system in terms of the four pillars and the associated capabilities described above. A major advantage of this modeling approach is that the model of a system contains model elements that are defined once in the model, but can appear on zero, one, or many diagrams. This provides a flexible means to present multiple views of the same system that are self consistent.

Many of these diagram types are used to reflect specific requirements, design, and implementation views of the system in alignment with ISO 42010 [13] and RASDS [14]. Detailed information on SysML can be found in several books on this topic including 'A Practical Guide to SysML' [2].

System Example

In this section, a simplified *Spacecraft* and *Ground System* end-to-end system design example is introduced to provide the context for the layered interface modeling pattern. A critical system interface requirement is specified, along with some of the considerations for allocating this requirement to other subsystem interfaces. In Section 3, the layered interface modeling pattern is applied to a particular subsystem interface from this example, and is described in more detail. Model elements that are referenced in the text are shown in *italics*.

End-to-End System Design

An end-to-end view of the *Spacecraft* and *Ground System* example is shown in Figure 5. The overall system function is to provide the observed *Temperature Data* of the *Thermal Sources* to the *User*. The *Spacecraft* transforms *Thermal Emissions* from the *Thermal Sources* into *RF Signals* that are transmitted to the *Ground System*, and the *Ground System* transforms the *RF Signals* to *Temperature Data* which is provided to the *User*.

RASDS Connectivity View objects (physical components) and links (connectors).

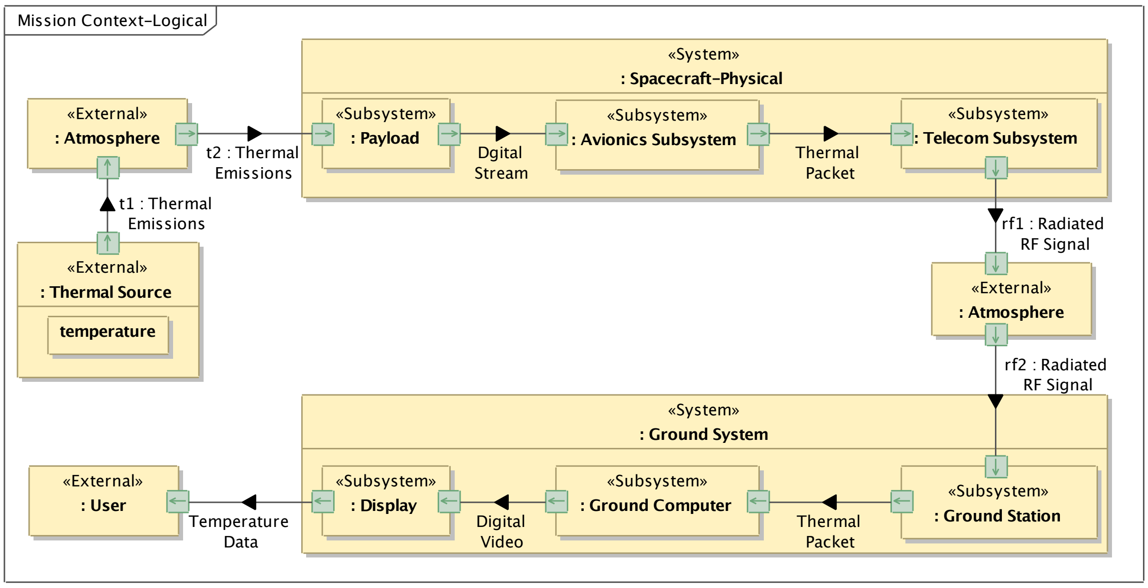


Figure 5: Example End-to-End View of Spacecraft and Ground System

Specifically, the observed *Thermal Sources* on the Earth's surface emit *Thermal Emissions,* which propagate through the Earth's *Atmosphere*. The *Spacecraft Payload* includes a *Sensor* that senses the *Thermal Emissions*. The *Sensor Signal* is processed by an *On-board Computer* in the *Avionics Subsystem* and converted to *Thermal Packets*. The *Telecom Subsystem* transforms the incoming *Thermal Packets* into space data link units, modulates the data, and transmits *RF Signals* through the Earth's *Atmosphere* to the *Ground System*. The *Ground System Receiver Subsystem* receives and demodulates the *RF Signal,* processes the space data link units, and extracts the *Thermal Packets*. The *Thermal Packets* are processed by the *Ground Computer* to derive the *Temperature Data* that can be stored as text files, MPEG videos, or other file formats. This data is also transformed to *Digital Video* to send to the *Display*, which is presented to the *User*.

System Data Interface Requirements and Allocation Approach

The basic top-level requirement for the end-to-end system as noted in the previous section is to provide *Temperature Data* of the *Thermal Sources* to the *User*. The *Temperature Data* provided to the *User* should be specified as an interface requirement for the end-to-end system. The *Temperature Data* is a logical abstraction of the physical signals provided directly to the *User*, which in this example, are photons emitted from the Display. The interface requirement should specify the *temperature* of the *Thermal Source* in units, such as degrees Celsius, and include the estimated time when the temperature was measured. There are many derived requirements to achieve the desired measurement quality and satisfy the user need including requirements related to sample rate, latency, range, accuracy, precision, reliability, and security. The requirement for this example may be stated as: *The end to end system shall provide estimated Temperature Data in degrees Celsius of the thermal sources located within the specified coverage area to the Users at the XYZ Facility every 4 hours with an accuracy of 1 °C over a temperature range from 0 °C to 300 °C*.

The data interface requirements must be satisfied by the end-to-end system. This in turn imposes requirements on all the system elements and associated interfaces that contribute to the end-to-end data flow. Latency is allocated to each system element in the data flow path. For example, accuracy may drive sensor resolution requirements. Precision and range requirements may drive the number of bits to be transmitted. The coverage area, accuracy, precision, and range requirement may drive the amount of data to be collected and transmitted, and the associated storage and downlink data rates. Reliability may drive selection of the communication protocols and the associated packet loss rate. Security confidentiality, integrity and availability may drive the need for encryption, access control, and firewalls.

Requirements are allocated to the system elements and their interfaces as the design process progresses. This process includes several design and implementation choices such as whether hardware or firmware is required to meet the performance goals, or does software suffice; and whether coax cables are sufficiently noise free or must fiber optics be used.

Interface Realization

The previous section introduced the end-to-end spacecraft system that is used as an example to illustrate the application of the layered interface modeling pattern. This system includes several components, both in the *Spacecraft* and the *Ground System* as shown in Figure 5. The process for moving from requirements to realization at each level of design involves further decomposing the system and its elements, defining the interfaces between them, and allocating the requirements to the next lower level elements. In order to ensure the system satisfies its requirements, the characteristics of the elements and their interfaces must be specified, designed, and verified.

In this section, the layered interface modeling pattern is applied to the *On-board Avionics Subsystem* and *Telecom Subsystem* interfaces shown in Figure 6. In this figure, the subsystem interfaces are shown as a single *Packet Port* that are connected by a connector that supports the exchange of *Thermal Packets*. The tilde symbol (~) on the *Packet Port* of the *Telecom*

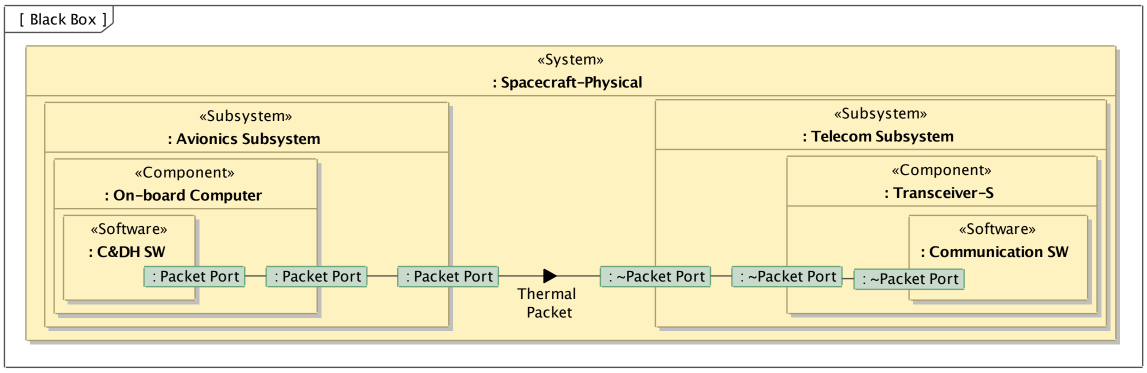


Figure 6: Overview of Component Interface (Black Box)

RASDS Connectivity View objects (physical components, HW & SW), Ports (interfaces) and links (connectors).

Subsystem indicates the port is conjugated to enable the flow direction to be reversed from out to in. Note that the subsystem ports are also connected to the ports on its internal components. This enables the subsystem interface to be specified as a black box that is realized by its internal components. As in the earlier example of the Audio Player to Amplifier interface in Figure 3, there may be several ways to realize this interface, which may have very different performance and behavioral characteristics. Furthermore, system-wide design decisions may constrain these choices.

Stack Definition

To fully specify the interfaces on a component, the protocol elements that make up the "stack" must be defined. Figure 7 shows the Avionics and Communication components from Figure 6 and defines the protocol stack for the *Packet Ports* on the two components. In more traditional spacecraft, this protocol stack might use MIL Std1553, LVDS, or even Spacewire. This example assumes the use of TCP/IP on-board to network together the sub-systems, and uses 1 Gigabit Ethernet and RJ-45 plugs. Although this sort of physical layer is not a typical spacecraft deployment, it is used for illustration purposes because it may be more familiar to many readers.

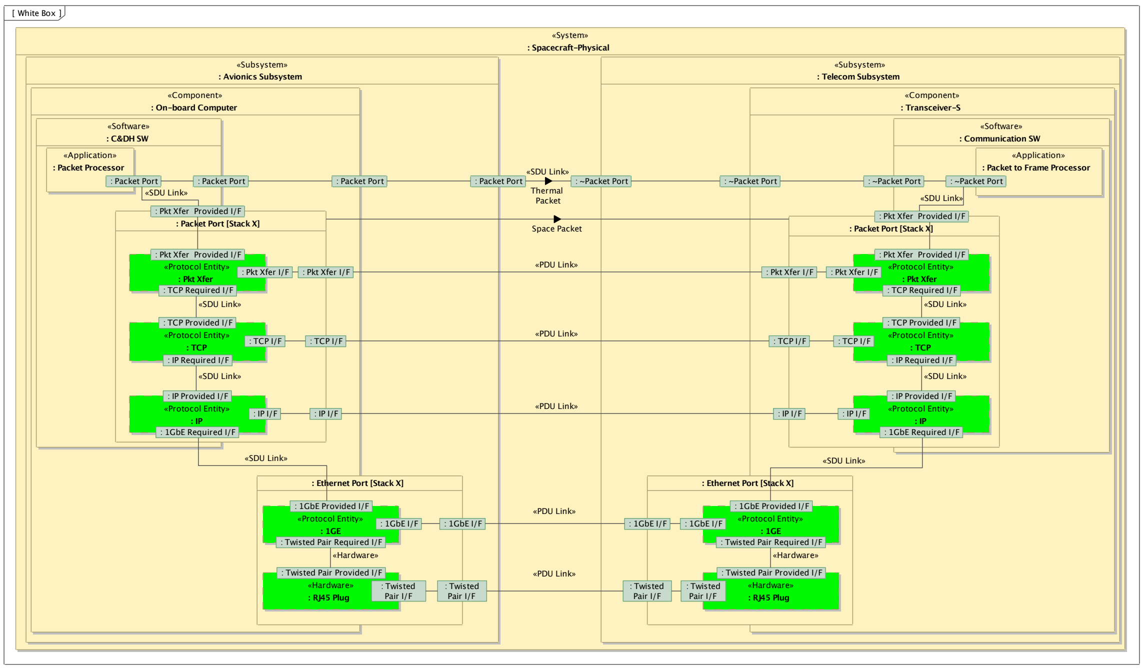


Figure 7. Protocol Stacks Inside Component Interface (White Box)

RASDS Communications (protocol) View, showing protocol stacks, with correspondence to Connectivity View objects (physical components, HW & SW), Ports (interfaces) and links (connectors).

The top level flow is still shown as *Thermal Packet*, but now the layers of the protocol stack are defined, and each layer has the «Protocol Entity» stereotype applied. The stack consists of the following:

1. Application protocol layer: packet transfer protocol, manages exchange of packet data between applications.
2. Transport layer: Transmission Control Protocol (TCP), provides end-to-end, once only, in order, complete delivery of data.
3. Network layer: Internet Protocol (IP), provides network layer routing over any number of intermediate network nodes.

4. Data link layer: 1 Gb Ethernet, provides data link layer services that may involve a fabric of switches and hubs.
5. Physical layer: twisted pair cable (Cat-5) and RJ-45 plug terminations.

The application above the protocol stack is responsible for processing the contents of the packets, and the protocol stack is responsible for transferring the packets. The top three protocol layers are responsible for the encoding and transfer of the data and are implemented in software. The bottom two layers are responsible for the physical and electrical connection, and are implemented in the computer.

The type of data flowing between the On-board Computer and the Transceiver-S in Figure 7 is Thermal Packets. That is, packets that contain the thermal measurements of interest. Stack X however, can accommodate any kind of Space Packet, including Thermal Packets. Thermal packets are, in the model, a specialization of Space Packets, see Figure 10. It's not shown in Figure 7 but many types of packets are routed through the same Stack X, not just Thermal Packets. This is an example of how the modeling can support re-use of the interface.

The data flows down the stack on one side and up the stack on the other. A protocol entity performs the appropriate behavior to support the transformation and exchange of data at that layer. Each layer is typically described by a single protocol specification that defines the behavior for that layer. Service Units (SDU's) are input to each layer from the layer above or below. The protocol behavior for a particular layer transforms its input SDU to an output SDU.

A protocol entity also interacts with its peer-level protocol entity at the same layer on the other side of the interface by exchanging Protocol Data Units (PDU's). The protocol behavior also specifies the transformation of the input SDU to an output PDU. The protocol entities are shown with dashed lines, which indicates that the protocol entities can be implemented elsewhere within the system.

Figure 8 extends concepts from our previous paper [5]. The figure shows elements of the protocol stack defined with stereotypes in a SysML profile that are used in the examples in this paper.

- A Component contains other Components, Protocol Entities and hosts Applications.
- Components perform Behavior.
- There are three types of Ports, Required I/F and Provided I/F (for SDU) and PDU I/F.
- SDU Links connect Provided I/F and Required I/F.
- PDU Links connect PDU I/Fs. Finally, PDU Data flows over PDU Links.
- Constraints govern PDU Links and SDU Links.

Each port on a system component has an interface binding signature that describes each layer of the protocol stack that interfaces with the peer interfacing component. Depending upon the nature of the component, the interface binding signature may have multiple layers, each with its own internal protocol stack, but there is always a physical layer and a link layer for communications. And, in some circumstances such as space communication, there may also be sub-layers. For example, in the Consultative Committee for Space Data Systems (CCSDS), the link layer is defined to include both data link and error encoding, and the physical layer is defined to include modulation and free space radiation.

Specifying subsystem and component interfaces by their interface binding signatures allows the design more flexibility. As technology evolves, the component design may also evolve from a set of elements built from discrete components to elements based upon FPGA firmware, to a single integrated package of software running on a high performance COTS CPU.

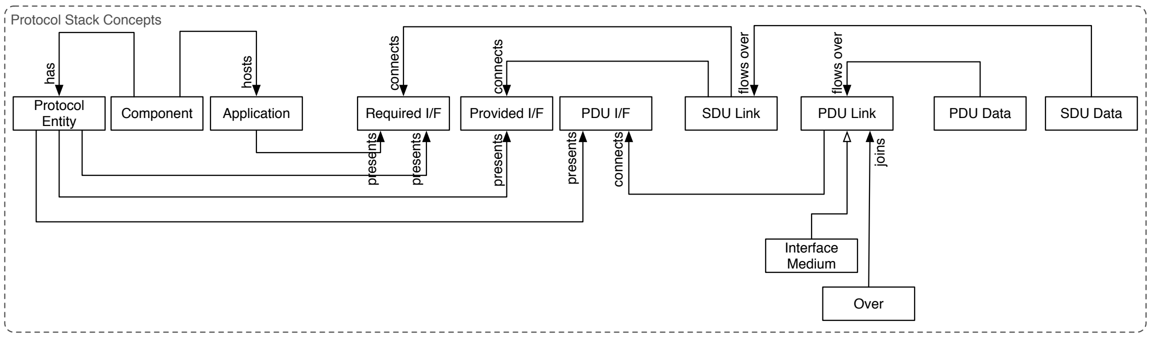


Figure 8. Concrete Protocol Stack Interface Concepts

There are two related, but somewhat disjoint statements about how protocol stacks operate:

1. The behavior of each protocol entity is carefully specified at each layer by describing how the two peer protocol entities in each interfacing component behave. This involves careful definition of the PDUs and behaviors within the layer.
2. The data does not flow directly between peer entities; it actually flows down one stack, across the physical connection, and up the other stack. The SDU interfaces between layered protocol entities are only abstractly defined.

One consequence of this approach is that the implementation details in two connected, but interoperable, components that share an interface may be entirely different. They may use different languages, run on different operating systems, and even allocate functionality very differently. However, they will interoperate as long as the protocol specifications are implemented faithfully.

The functions at each layer may be implemented in one component or they may be allocated to different components, depending on design choices. For example, in many space systems, the RF, modulation, encoding, and data link functions may be allocated to separate components. The *Transceiver* shown in Figure 7 includes link layer, encoding and modulation functions. However, the *Transceiver* can be implemented as three separate components, a link layer processor, an encoder, and a modulator. In this case, each of these sub-components also have their own top level interfaces, passing application data, data link frames, and encoded data blocks. In a recursive fashion, the interfaces themselves each have an interface binding signature that typically include one or more layers.

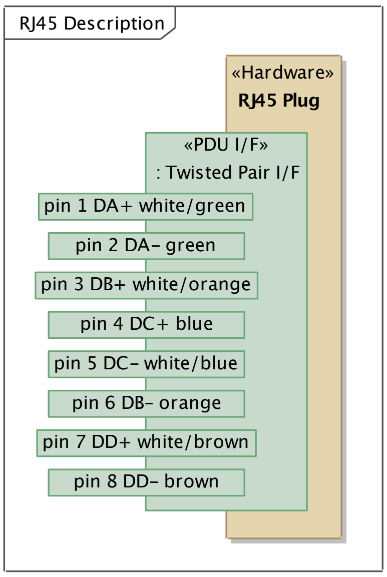


Figure 9. RJ45 Pinout

Each of the protocol entities in the stack may be left as abstract or further elaborated, as needed. The interface between IP and the 1 gigabit Ethernet data link may be important to define, along with the CSMA/CD behavior. The RJ45 plug and Cat5e Ethernet cable may be elaborated to show its electrical pin out as shown in Figure 9. The plug specification may include electrical

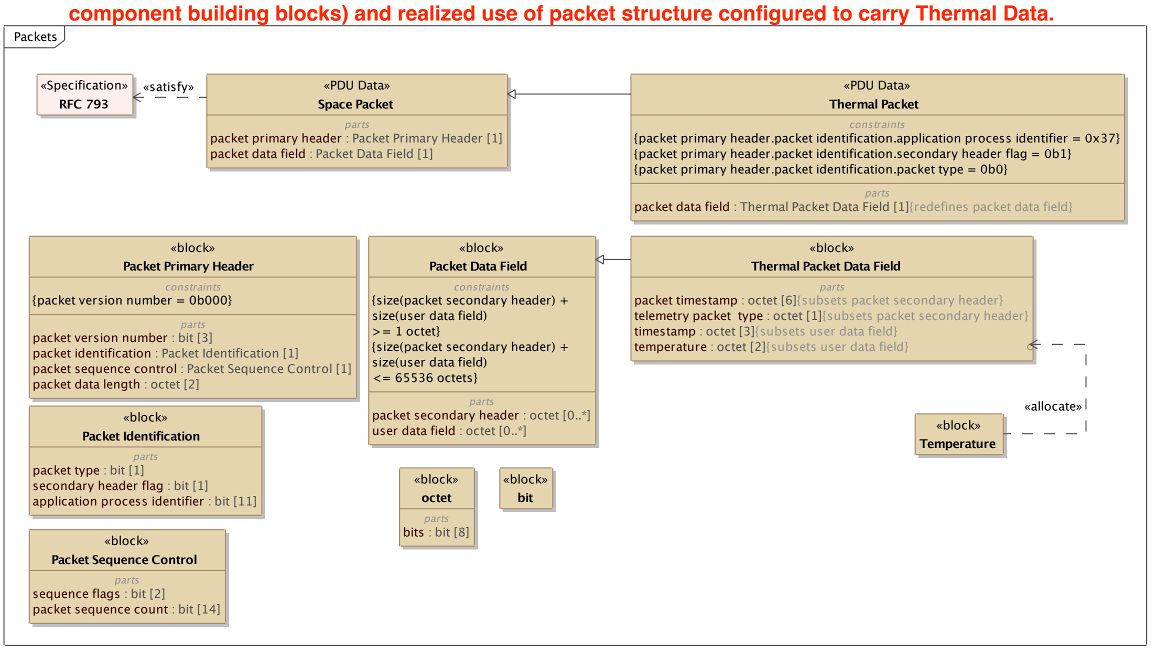
RASDS Connectivity View object (physical component, HW interface view), showing physical detail of connector.

and mechanical properties, such as impedance. Each pin can be specified individually, with whatever additional information is needed.

Packet Data Structures

The data objects that are exchanged (i.e. data flows) must have a well defined data structure down to the octet and bit level, and have well defined relationship with other data objects. The data structure definitions are constructed using defined building blocks. All data structures, including PDUs and SDUs at each layer, must be defined in unambiguous terms to ensure interoperable exchange of information between applications. Figure 10 shows the data structure definition for a CCSDS Space Packet [16], which is the highest-level data structure for Stack X shown in Figure 7. It also shows the Thermal Packet that flows on Figure 7. The Thermal Packet is contained in the CCSDS Space Packet. This Packet Data structure may be used to carry many different types of application data, and it may also carry application layer signal information as well as provide limited functions for data assembly and/or fragmentation.

RASDS Information Viewpoint (data) View, showing abstract definitions of data structures (packet & component building blocks) and realized use of packet structure configured to carry Thermal Data.



Packets
«Specification» RFC 793
«satisfy»
«PDU Data» Space Packet
parts
packet primary header : Packet Primary Header [1]
packet data field : Packet Data Field [1]
«PDU Data» Thermal Packet
constraints
{packet primary header.packet identification.application process identifier = 0x37}
{packet primary header.packet identification.secondary header flag = 0b1}
{packet primary header.packet identification.packet type = 0b0}
parts
packet data field : Thermal Packet Data Field [1]{redefines packet data field}
«block» Packet Primary Header
constraints
{packet version number = 0b000}
parts
packet version number : bit [3]
packet identification : Packet Identification [1]
packet sequence control : Packet Sequence Control [1]
packet data length : octet [2]
«block» Packet Data Field
constraints
{size(packet secondary header) + size(user data field) >= 1 octet}
{size(packet secondary header) + size(user data field) <= 65536 octets}
parts
packet secondary header : octet [0..*]
user data field : octet [0..*]
«block» Thermal Packet Data Field
parts
packet timestamp : octet [6]{subsets packet secondary header}
telemetry packet type : octet [1]{subsets packet secondary header}
timestamp : octet [3]{subsets user data field}
temperature : octet [2]{subsets user data field}
«allocate»
«block» Temperature
«block» Packet Identification
parts
packet type : bit [1]
secondary header flag : bit [1]
application process identifier : bit [11]
«block» octet
parts
bits : bit [8]
«block» bit
«block» Packet Sequence Control
parts
sequence flags : bit [2]
packet sequence count : bit [14]

Figure 10. Packet Data Structure definition

The application data at the packet transfer layer may have explicit structure known to the application, but that data is treated as bits that are only meaningful to the layer above. The data structure in this example is the *Space Packet*, which is defined as a data structure with two parts, the *Packet Primary Header* and the *Packet Data Field*. Both of these are defined in a way that promotes re-use. The example shows the specialization of *Packet Data* to carry typical thermal data in the *Thermal Packet*, redefining the generic *Packet Data* as *Thermal Packet Data Field* that specifies the structure of the specific application data. Other packet data types may be specified in a similar way.

Protocol Entity Behavior

Accurately characterizing the behavior and performance of each interface requires an understanding of the protocol stack, and understanding the stack requires an understanding of the behavior of the protocol entities at each layer. This section and the next provide a method for describing that behavior.

Within the stack, the SDU at each layer (N) is a sequence of octets that is provided by the upper layer (N+1) transformed by the (N) layer protocol and then passed to the lower layer (N-1). It is a function of the upper layer to send the SDU in a form that is acceptable to the (N) layer service interface. The (N) layer may transform that SDU in a variety of ways, including cutting it into smaller pieces, aggregating it into larger units, or performing a transformation to encode or encrypt the SDU. It is a function of the (N) layer to send the (N) SDU it constructs to the (N-1) layer service interface in a form that is acceptable to that layer.

Figure 11 shows one of the protocol entities, the Transmission Control Protocol (TCP), that is part of the stack in Figure 7. The TCP protocol entity has ports like the other protocol stack elements. Each protocol entity has three ports, the interface that provide services to the upper (N+1) layer, the interface that requires services of the lower (N-1) layer, and the interface with the peer protocol entity at the same layer:

1. Provided service port: the services offered to any upper layer (N+1) protocol, defined as an abstract service and using a layer N Service Data Unit (SDU)
2. Required service port: the services required from any lower layer (N-1) protocol, defined as an abstract service and using a layer N-1 Service Data Unit (SDU)
3. Peer protocol port: the port that enables the protocol entity to interact with its peer entity at the same layer, defined by the protocol specification and using the layer N Protocol Data Units (PDU)

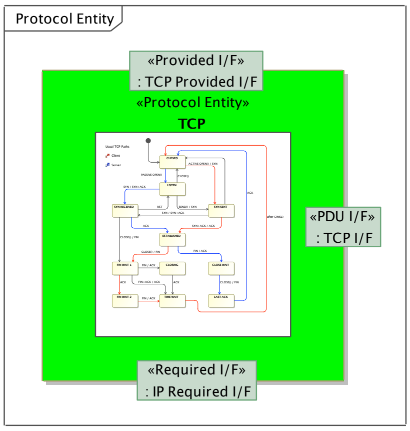


Figure 11. Protocol Entity (Black Box)

RASDS Communications (protocol) Viewpoint, showing protocol stack element and interfaces

There may also be a separate control or management interface within the protocol, or via a separate port on the component.

The provided service interface accepts layer (N) Service Data Units (SDU) from the upper (N+1) layer, and it is the upper layer entity's job to match the implementation characteristics of the layer (N) provided interface. Similarly, the interface to the lower layer (N-1) protocol entity must provide (N-1) SDUs in the form that entity expects. Within each protocol entity is a transformation engine that accepts (N) SDUs, creates (N) PDUs that contain all or part of each (N) SDU, and then forms (N-1) SDUs for the lower layer. This sending side process, of course, works in reverse in the peer protocol stack on the receiving end.

One (or more) state machines and/or activity diagrams may be used within the protocol specification to define the protocol entity's behavior. While some protocol specifications will contain carefully constructed state machines, or state tables, some of them use English prose to specify the behavior. The SysML modeling approach used here provides explicit state machine, sequence, and activity diagrams to describe the behavior.

One or more state machines are needed to describe peer level protocol behavior; and how the protocol entity responds when PDUs arrive, and what PDUs are sent. This behavior may involve establishing a connection, authentication, performing mono- or bi- directional data exchanges, handling reliability & error conditions (re-transmission), sending and responding to quality of service (QoS) signals, and other behavior. There may also be data transformation behavior within the protocol entity that can be described using a separate state machine or functional model (activity diagram). This describes the transformation of (N)

SDUs into (N-1) SDUs, which may require data fragmentation, re-assembly, caching, and even data transformation to encode or encrypt.

Figure 12 shows the canonical State Machine for TCP connection establishment and tear down. This diagram combines the state transitions for both the sender and the receiver. Either TCP entity may send and/or receive. In support of the sender / receiver roles, it is useful to think of one TCP entity as the server and the other as the client. From this point of view, the server leaves the *Closed* state using the blue path, performs a *passive open*, and enters the *Listen* state. When a client is ready to communicate, it exits the *Closed* state via the red path, performs an *active open*, and sends a *SYN PDU*. The PDU exchanges continue until the connection is Established, at which point the two peer entities may exchange data in either direction. The bottom part of Figure 12 describes the process for closing the connection and returning to the *Closed* state. A very similar diagram is included in RFC 793 [8] that defines the TCP protocol.

RASDS Communications (protocol) Viewpoint, showing protocol behavior view within a protocol entity using state machine diagram.

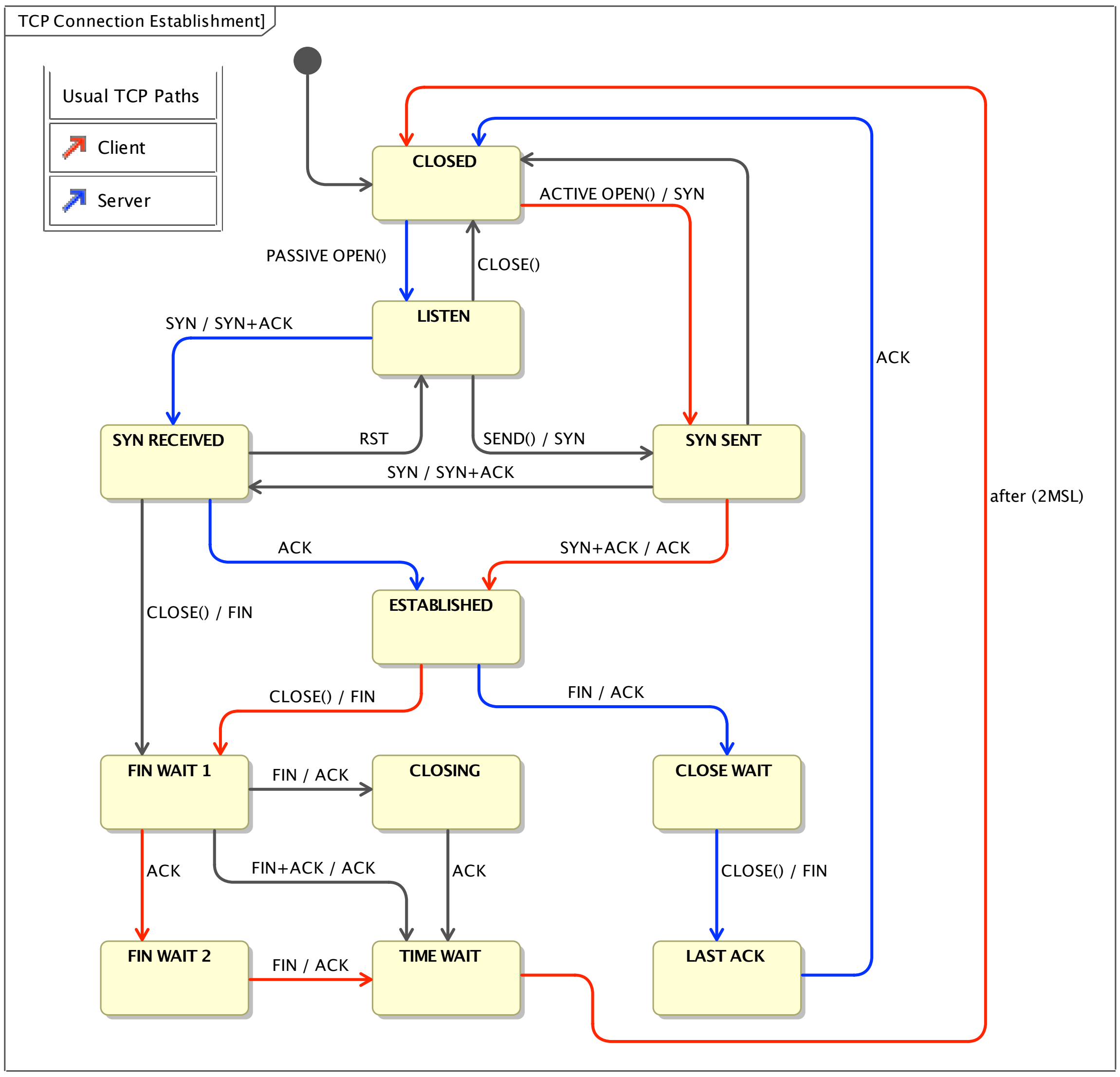
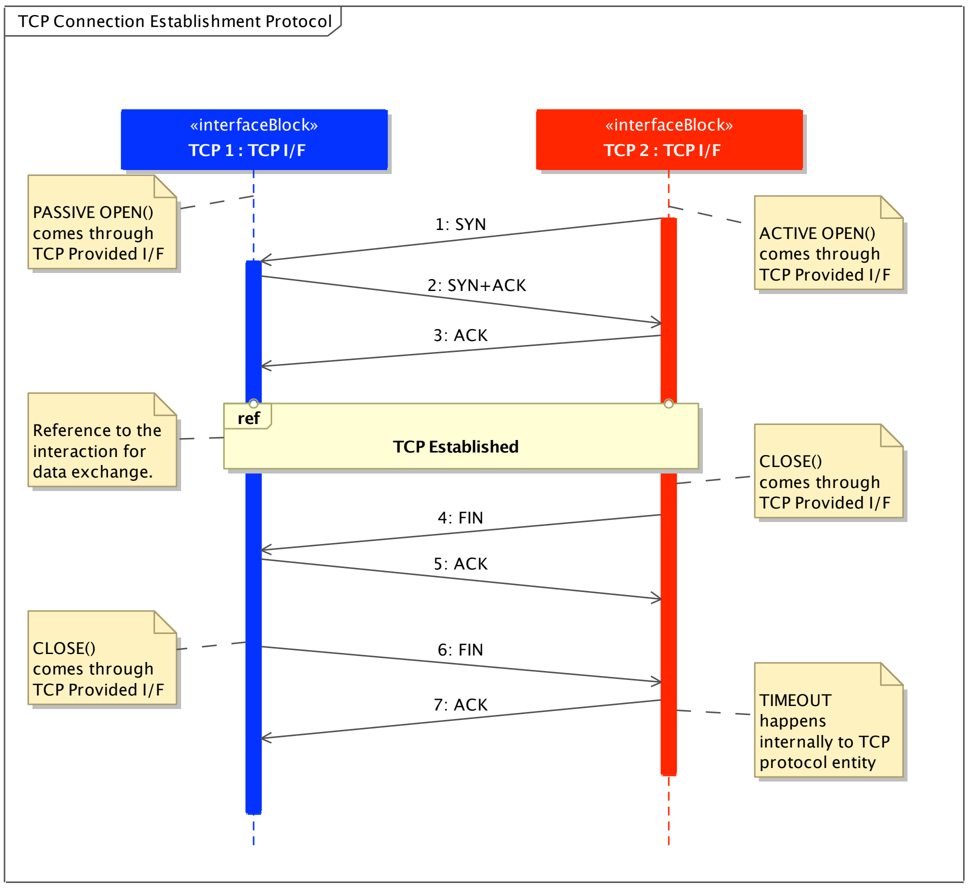


Figure 12. TCP Protocol state machine – connection establishment

Figure 13 is a sequence diagram that depicts a part of the PDU interchange between the client and the server, showing the PDUs sent by the client (TCP 2) in red, and PDUs sent by the server (TCP 1) in blue. As mentioned, the server does a *passive open* and the client does an

RASDS Communications (protocol) Viewpoint, showing protocol exchange view between peer protocol entities using sequence diagram.

active open. The top part of the diagram shows the exchanges to achieve the *Established* state. Those on the bottom are the exchanges to return to the *Close* state.



TCP Connection Establishment Protocol
«interfaceBlock» TCP 1 : TCP I/F
«interfaceBlock» TCP 2 : TCP I/F
PASSIVE OPEN() comes through TCP Provided I/F
ACTIVE OPEN() comes through TCP Provided I/F
1: SYN
2: SYN+ACK
3: ACK
ref
TCP Established
Reference to the interaction for data exchange.
CLOSE() comes through TCP Provided I/F
4: FIN
5: ACK
CLOSE() comes through TCP Provided I/F
6: FIN
7: ACK
TIMEOUT happens internally to TCP protocol entity

Figure 13. TCP Protocol connection establishment sequence diagram

Much of the behavior of a protocol entity behavior may be captured within the state machines, but the dynamics of the interactions between the two cooperating state machines can be further described using a sequence diagram. The sequence diagram can show both the interchanges of PDUs and also the timing relationships as the protocol entity interacts with its peer entity at the same level. Particularly for space data link protocols, where there may be long round trip light time delays (tens of minutes to tens of hours), understanding the timing dynamics of the protocol becomes very important. One approach to dealing with long delays is to use a networking approach like Delay/Disruption Tolerant Networking (DTN) that uses the delay tolerant Bundle Protocol [12] rather than TCP and IP. Understanding the dynamics of interactions is important when there is a high bandwidth / delay product (> 10^7 bits). Sequence diagrams that describe these interactions and timing considerations are useful in understanding protocol behavior in the face of data errors, data loss, weather based channel fades, and other conditions.

Once the connection is established, the TCP operates to provide reliable end-to-end exchange of a stream of bytes, in order, once only, and without omission. That behavior takes place within the *Established* state in Figure 12. In RFC 793, this description is many pages of clear, but rather dense, prose. Specifying this behavior as behavioral models facilitates understanding, and the translation of this behavior into code. It is even possible to automatically transform well specified state machines directly into executing code. [17]

The behavior of TCP to accomplish these reliable exchanges is complex and it must handle start up, shut down, retransmission, and error cases. It is a testament to the quality of the original spec that it has persisted in essentially an unchanged form since 1981, leaving aside extensions like SACK and RENO.

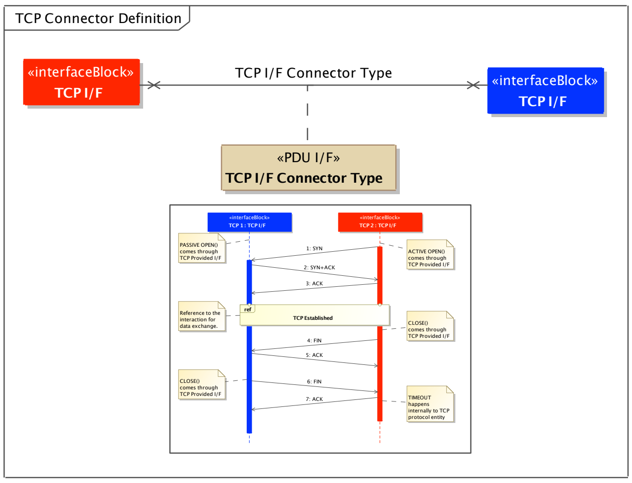

TCP Connector Definition
«interfaceBlock» TCP I/F
TCP I/F Connector Type
«interfaceBlock» TCP I/F
«PDU I/F» TCP I/F Connector Type
«interfaceBlock» TCP 1 : TCP I/F
«interfaceBlock» TCP 2 : TCP I/F
PASSIVE OPEN() comes through TCP Provided I/F
1: SYN
2: SYN+ACK
3: ACK
ACTIVE OPEN() comes through TCP Provided I/F
Reference to the interaction for data exchange.
ref
TCP Established
CLOSE() comes through TCP Provided I/F
4: FIN
5: ACK
CLOSE() comes through TCP Provided I/F
6: FIN
7: ACK
TIMEOUT happens internally to TCP protocol entity

Figure 14. Association block and Sequence diagram specifying relationship between port definitions (i.e., types)

Figure 14 shows the sequence diagram from Figure 13 which constrains the interaction between the client and server, which are peer-level entities on each side of the connection.

Figure 15 provides a structured description of a part of the *TCP Established* state behavior. This describes the behavior of the sending protocol entity when it receives *SEND Call* on its (N+1) service interface. The behavior is described using an activity diagram. The figure re-casts in SysML a figure from a paper modeling TCP and the RENO congestion control extensions using EFSM/SDL [9]. The diagram shows both the calculations done within the protocol entity to manage the sliding acknowledgement window (rounded ovals) and the protocol behavior to send a PDU with a segment of data and with the *SND*, *ACK*, and *CTL* signal field settings (right facing arrow).

The defined behavior and PDU exchanges in each state are defined in Figure 12. In the cited paper the full description of the protocol behavior in EFSM is documented in many pages.

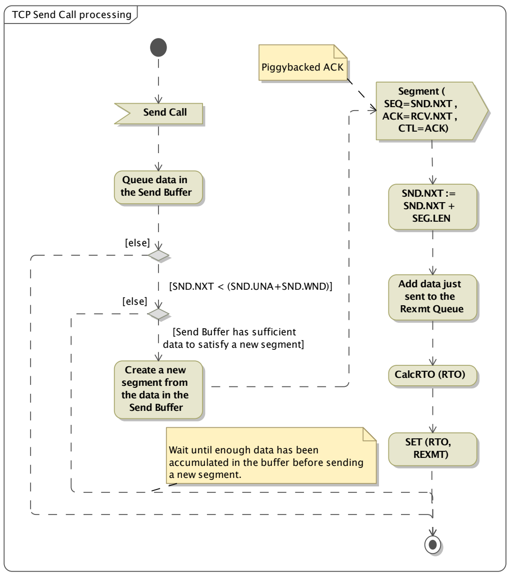

TCP Send Call processing
Piggybacked ACK
Send Call
Segment (SEQ=SND.NXT , ACK=RCV.NXT , CTL=ACK)
Queue data in the Send Buffer
SND.NXT := SND.NXT + SEG.LEN
[else]
[SND.NXT < (SND.UNA+SND.WND)]
Add data just sent to the Rexmt Queue
[else]
[Send Buffer has sufficient data to satisfy a new segment]
Create a new segment from the data in the Send Buffer
CalcRTO (RTO)
Wait until enough data has been accumulated in the buffer before sending a new segment.
SET (RTO, REXMT)

Figure 15. TCP Send Call activity diagram

RASDS Communications (protocol) Viewpoint, showing protocol behavior view within a protocol entitiy using activity diagram.

Compliance with RFC 793

Figure 16 gives an overview of all the elements in the model that must comply with RFC 793. The same pattern will be found for any other protocol as well. RFC 793 governs the PDU link at the TCP layer, the two TCP Protocol Entities, the two SDU Links above, and all six Interfaces to which those links connect. This compliance includes data, structure, connections and behavior, as illustrated by the state machine, activity and sequence diagrams. Compliance is shown by the Satisfies notation on the diagram both on the Components (white box) and the protocol entities (black box).

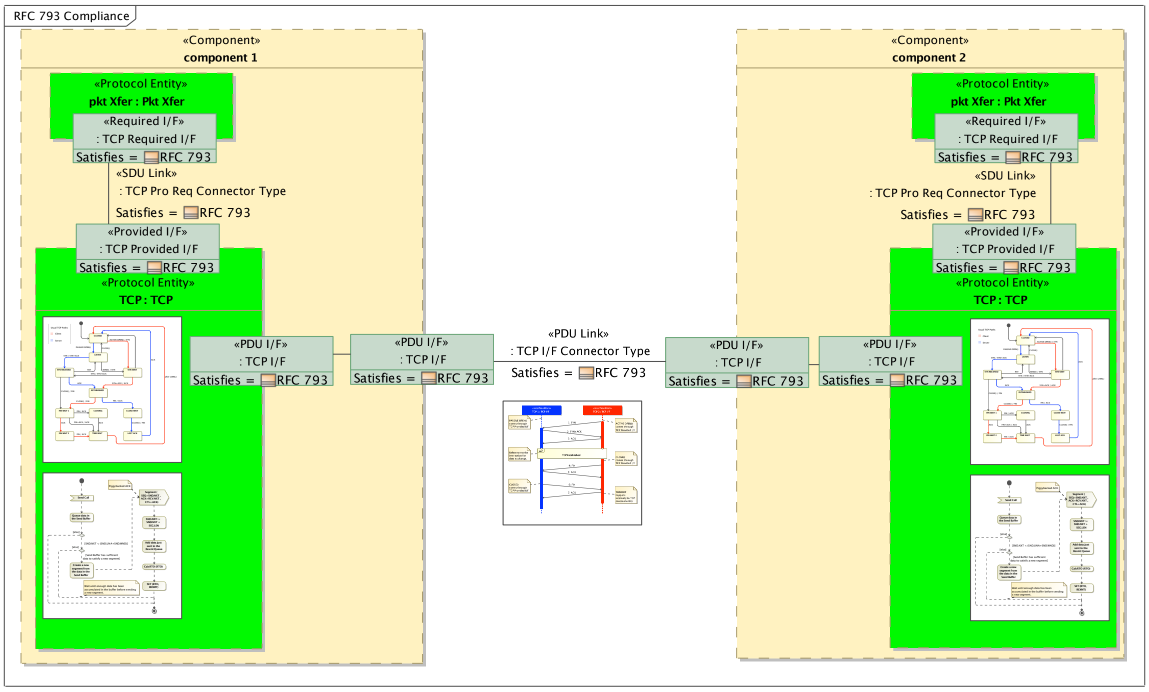


Figure 16. Compliance with RFC 793

Related Work

This pattern is the result of work performed on several previous tasks at the Jet Propulsion Laboratory. The pattern was first developed and applied as part of the Space Communication and Navigation (SCaN) Integrated Network Interface Definition Trade Study and the related SCaN Network Integration Project (SNIP). It was used to describe and document each of the standard interfaces to help unify the three separate Earth to Space communications networks run by NASA. These standard interfaces allowed encapsulation of different implementations, providing common external interfaces for end users, regardless of which communications network they chose, and common internal interfaces as integration points. The interface bindings were described from application layer down to network layer and tied to the individual protocol layer specifications. This approach provided accurate models of all of the major external and internal interfaces.

The pattern was also applied to the Ground Data System of the Exploration Flight Test 1 (EFT-1) project. The purpose here was to describe the flow of information across the ground network supporting the mission. Two levels of abstraction were used. The top layer described the flow from source to destination in a single step, and the second layer described the connections between the routers, switches, firewalls and servers. Constraints were added to describe the path of the first over the second.

There has been other earlier work to model interfaces in SysML, some of which started to model similar layer interface concepts. Robert Karban developed and applied interface and protocol stack patterns to model software, electrical, optical, and mechanical interfaces while at the European Southern Observatory [19]. Mark McKelvin applied layered interface patterns to electrical interface design [21] [22]. Maddalena Jackson wrote about using a layered interface concept to describe data flows in support of human space flight.

Applying the Pattern to Other Types of Interfaces

This paper largely focuses on communications interfaces, since dealing with the complexities of a full specification of such interfaces was a driver for developing the layered interface pattern. However, as stated in The Interface Challenge section, systems interfaces can be very diverse and involve many different technical domains including electrical, mechanical, thermal, software, user, and others. A question to be explored here is whether this pattern can be leveraged to model other kinds of interfaces.

One key observation is that many interfaces between elements exhibit a set of characteristics that may be modeled as a stack with defined functionality. An earlier paper [5] introduced the concept of four abstract layers: message, encoding, signal, physical. These examples that were used correspond to data exchanges using various protocols, to send signals across the interface media such as a cable or free space, and the physical connection to the interface media. This interface modeling pattern, however, may be generalized to model interface with distinctly different characteristics.

Communication Interfaces

Applying this layered pattern to specify communication interfaces has been presented above. The communication interfaces come in a wide variety of forms and may include only two protocol layers, or many of them (*encoding*). The sorts of *signals* used may also vary widely, depending upon the *physical* media being used in the communication path (free space RF or optical, fiber, copper wire, and maybe, in some future, quantum entanglement).

Intermediate system components, such as routers or switches, may only include two or three layers (up to link or network layers), and other components, such as gateways, may include protocol transformation behavior as well. All of these may be modeled by applying this pattern. Furthermore, end-to-end-performance may be modeled by defining implementation specific performance characteristics for each of the physical interconnects and, if necessary, modeling the performance of SDU transformations within the stack to account for these time and resource consuming processes.

Other Interfaces

This modeling pattern may be extended to model interfaces that are constrained by physical laws, such as forces, torques, momentum, and energy. Modelica is a modeling language that simulates physical interactions, and expresses the constraints in terms of conservation laws. The model of the physical layers of the stack can be augmented to reflect these constraints.

An example for RF antenna, gimbal and inertial effects is as follows. The RF antenna may be body mounted on a spacecraft, or it may be on a gimbal. If it is on a gimbal there will be control and power interfaces, as in the previous example, but there will also be inertial effects on the rest of the spacecraft. The system will have to react to this, using counteracting forces driven managed by control loops. These will have their own interfaces for control and power and their own sets of constraints.

Application of the modeling pattern to user interfaces may be an interesting area for future exploration. Humans gather information using their five senses, and then decode the information using their nervous system and brain. When describing user interfaces, the human "stack" and related aspects of the end-to-end system flow are often abstracted away, there may be cases where elaborating this part of the model may provide useful insights.

Summary

The application of the pattern to communication interfaces has been described to help guide the consistent and clear specification and design of end-to-end system interfaces. It provides a framework for modeling system and component interfaces at successive levels of detail as the design progresses. It helps to address the complexity of each interface in terms of how data is encoded in messages and signals, the rules that govern their exchange, and how they are physically sent from sender to receiver. This pattern defines how to model these interfaces and to document conformance to standards.

The pattern provides the modeler the ability to accurately describe complex interfaces at whatever level of detail is useful. Interfaces may be left abstract at the "communicated data" layer if documenting end-to-end connectivity is all that is required. The interfaces can also be documented down to the physical layer, including performance characteristics, such that throughput and latency may be characterized. External and internal interfaces may be given the same treatment using the same pattern.

The effective specification and design of external and internal interfaces is a critical aspect of any system development process. The number, diversity and complexity of interfaces contributes to the interface specification and design challenge. A model-based approach can help address this challenge over more traditional document-based approaches by enhancing consistency, precision, traceability, conformance to standards, and reuse.

This paper uses a representative Spacecraft and Ground System to illustrate how a critical end-to-end system interface is specified using a model-based approach with SysML. It then presents an application of a layered interface modeling pattern to realize a Spacecraft subsystem interface and help manage the inherent complexity. The pattern leverages layered interface concepts to model each side of an interface as a stack of protocol entities with distinct functionality. Inputs flow down the stack on one side of the interface, across a physical medium, and up the stack on the other side of the interface.

The pattern specifies how to model a typical protocol entity, and its behavior to transform its inputs data to outputs at the next layer of the stack, and its interaction with a peer level protocol entity at each layer of the stack. It shows how to model data that flows through the stack as a logical abstraction, data encoded in bits and bytes, data encoded in signals such as electrical, RF, and optical signals. It also discusses how to model the physical connection to an interface medium such as a cable or free space.

Like any effective modeling effort, it is essential to scope the model to address the modeling objectives. This will result in emphasizing particular aspects of the interface for a given project and lifecycle phase. This pattern is intended to support such adaptation, and can be selectively and incrementally applied to meet a project's needs. Early in the development, the emphasis may be to create abstract models of the interface specification, and as the design progresses, the model may include additional design detail to address protocol, deployment, software, electrical, and mechanical design concerns. The details may be captured directly in the model, or refer to detailed interface information captured in other tools. Understanding the layered interface pattern can assist the team in determining an effective strategy for capturing this critical information to meet the needs of the project.

Although the pattern is illustrated for a communications interface, the application spans system, software, electrical, and mechanical interfaces. Future work can explore how to leverage this pattern for other kinds of interfaces that involve many technology domains and different engineering disciplines.

Part of the research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

[1] Object Management Group. OMG Systems Modeling Language (OMG SysML™). V1.4. Available at: http://www.omg.org/spec/SysML/.

[2] A Practical Guide to SysML, The Systems Modeling Language, Third Edition by Sanford Friedenthal, Alan Moore, and Rick Steiner, Morgan Kaufmann, 2014

[3] Estefan Jeff A., Survey of Model-Based Systems Engineering (MBSE) Methodologies, Rev B INCOSE Technical Publication, Document No. INCOSE-TD-2007-003-01. San Diego, CA: International Council on Systems Engineering; June 10, 2008.

[4] Interface Control Document NASA 932 C-9BAircraft Operations Division February 2011. http://jsc-aircraft-ops.jsc.nasa.gov/Reduced_Gravity/docs/AOD_33912.pdf

[5] Shames, Peter M, Sarrel, Marc A, A modeling pattern for layered system interfaces, 25th Annual INCOSE International Symposium (IS2015), Seattle, WA, July 13 – 16, 2015

[6] Universal Serial Bus Specification (revision 2) http://sdphca.ucsd.edu/Lab_Equip_Manuals/usb_20.pdf

[7] Guide to the Systems Engineering Body of Knowledge (SEBoK) http://sebokwiki.org/wiki/Guide_to_the_Systems_Engineering_Body_of_Knowledge_(SEBoK)

[8] Postel J., "Transmission Control Protocol," RFC 793, September 1981.

[9] Zaghal, R, Khan, J, EFSM/SDL modeling of the original TCP standard (RFC793) and the Congestion Control Mechanism of TCP Reno, Kent State University report, TR2005-07-22-tcp-EFSM.pdf, 2005

[10] Information technology - Open Systems, Basic Reference Model, ISO/IEC 7498-1, revised June, 1996

[11] OMG QUDV, Quantities, Units, Dimensions and Values, http://www.omgwiki.org/OMGSysML/doku.php?id=sysml-qudv:qudv_owl

[12] Scott, K, Burleigh, S, "Bundle Protocol Specification", RFC 5050, Nov 2007

[13] Systems and software engineering — Recommended practice for architectural description of software-intensive systems, ISO/IEC 42010, July 2007, revised 2011

[14] Reference Architecture for Space Data Systems (RASDS), CCSDS 311.0-M-1, Sept 2008

[15] Rasmussen, R, et al, An Architectural Pattern for Goal-Based Control, IEEE Aerospace Conference. Big Sky, MT. March 2008

[16] Consultative Committee for Space Data Systems, CCSDS Space Packet Protocol, CCSDS 133.0-B-1c2, Sept 2010

[17] Wagstaff, et al, Automatic Code Generation for Instrument Flight Software, citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.126.548

[18] Jackson, et al., “Architecting the Human Space Flight Program with Systems Modeling Language (SysML)”, Infotech 2012, AIAA 2012-2556.

[19] Karban, et al., “MBSE Initiative – SE2 Challenge Team, Cookbook for MBSE with SysML, Issue 1, INCOSE, 2011.

[20] Leveson, “The Role of Software in Spacecraft Accidents,” AIAA Journal of Spacecraft and Rockets, to be published.

[21] Mark L. McKelvin, Jr.,Robert Castillo, Kevin Bonanne, Michael Bonnici, Brian Cox, Corrina Gibson, Juan P. Leon, Jose Gomez-Mustafa, Alejandro Jimenez, and Azad M. Madni. “A Principled Approach to the Specification of System Architectures for Space Missions”, In Procs. of the AIAA Space Conference, Anaheim, CA, 2015.

[22] Mark L. McKelvin, Jr. and Alejandro Jimenez. “Specification and Design of Electrical Flight System Architectures with SysML”, AIAA Infotech@Aerospace, Garden Grove, CA, June 2012.

Biography

Peter Shames has been engaged in the process of turning computers into useful tools for scientists for the bulk of his professional career. His specific expertise is architecting large-scale space data systems, including space communications protocols and standards. Peter manages JPL's Data Systems Standards Program in the Interplanetary Network Directorate (IND). He is Director of the System Engineering Area for Consultative Committee for Space Data System (CCSDS). For CCSDS he was lead editor of the CCSDS Reference Architecture for Space Data Systems (RASDS, CCSDS 311.0-M-1) and Space Communications Cross Support Architecture (SCCS-ADD, CCSDS 901.0-G-1).

Marc Sarrel is a systems engineer in JPL's Mission Control Systems section. For the past five years, he has applied Model Based Systems Engineering to various system engineering tasks in the space-flight ground-systems domain. He has worked on the Spitzer and Cassini missions as a Mission Operations System Engineer and a Ground Data Systems Engineer, and has written ground processing software. He has a master’s degree in Computer and Information Science from The Ohio State University, a bachelor’s in Computer Science from Washington University in St. Louis, and has worked at JPL for twenty-five years.

Sanford Friedenthal is an independent consultant and industry leader in model-based systems engineering. Previously, as a Lockheed Martin Fellow, he led the corporate engineering effort to enable Model-Based Systems Development across the company, where he was responsible for developing and implementing strategies to institutionalize the practice of MBSD across the company, and provide MBSE support to programs. He chairs the INCOSE MBSE Initiative and other industry modeling efforts, and is co-author of ‘A Practical Guide to SysML.’