Modeling systems-of-systems interfaces with SysML

Peter M. Shames[1] and Marc A. Sarrel.[2]
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 91109

and

Sanford A. Friedenthal[3]
SAF Consulting, Affiliate, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109

Space data systems are inherently complex. They are systems-of-systems, typically composed of spacecraft and mission operations systems (MOS) belonging to one (or more) organizations, and multi-mission communication assets belonging to other organizations. In many cases, the spacecraft contain sub-systems and instruments provided by different organizations, and MOS systems that may be developed and operated by other organizations. The point of greatest leverage in system architecting is at the interfaces. We have developed a set of methods for using SysML to model systems-of-systems and their interfaces. This paper describes how to apply this method to space data systems at a variety of levels of detail, from abstract systems and subsystems down to hardware and software components, including the details of their interfaces and protocol designs.

I. Introduction

SPACE data systems are typically composed of spacecraft and mission operations systems (MOS) belonging to one (or more) organizations, and they often use multi-mission communication assets belonging to other organizations. These End-to-End data systems are inherently systems-of-systems composed of different elements, some of which may be free-flying in very remote locations. Well understood interfaces among these elements are essential.

To quote Mark Maier, from *Architecting Principles for Systems-of-Systems* [1].

"The greatest leverage in system architecting is at the interfaces. The greatest dangers are also at the interfaces."

The simple "boxes and lines" diagrams that are often employed in describing systems do not do justice to defining these interfaces, nor do Interface Control Documents (ICD) consisting of lists of identified standards. Even Model-Based Systems Engineering (MBSE) techniques and tools, if not carefully applied, may yield poorly defined interfaces. The question then is how to more accurately and completely specify and design these systems and interfaces to address a diverse set of stakeholder concerns?

This paper briefly introduces the problem domain, the End-to-End description of space data systems and their composition and connectivity. It starts with a top-level view that describes the overall mission context, including the major types of elements, nodes, systems, and subsystems. It then introduces the essential considerations for modeling these systems-of-systems and the interfaces that define their connection points. The essential features of these interfaces are introduced with some simple examples before diving into the details of modeling them with SysML [2].

The protocol stack layering and terminology that is used has been adapted from the ISO Basic Reference Model (BRM), ISO/IEC 7498 [3]. The modeling approach, using views and viewpoints, aligns with the ISO Recommended practice for architectural description of software-intensive systems, ISO/IEC 42010 [4] and the space data system modeling viewpoints define in the CCSDS Reference Architecture for Space Data Systems (RASDS) [5]. The technical details of using SysML to model interfaces are covered in more depth in another paper by the authors, to be published in INCOSE IS 2016 [6].

---

[1] Mgr, JPL Data System Standards Program, Interplanetary Network Directorate, MS 301-490.
[2] Systems Engineer, Mission Control Systems Section, MS 301-480.
[3] Affiliate, Engineering Development Office, MS 301-237.

At the most abstract level, just the data flows among the major components may be modeled. When needed, these interfaces may be elaborated to include protocol stacks with varying levels of detail that span from the application layer down to the physical layer. The protocol stack within an interface is modeled as a full port with an interface binding signature. This port contains parts that correspond to protocol entities at each layer of the stack. Each protocol entity interacts with its upper and lower layers in the stack, and with its peer level protocol entity of the interfacing component.

The method describes how each protocol entity and the different layers of the stack may be reused and applied where these types of ports and bindings are needed. Furthermore, the modeling methods also specify how to model protocol behavior. This may include modeling the behavior of each protocol entity (e.g., state machines), and the constraints on the interaction between protocol entities (e.g., sequence diagrams).

The method is intended to allow complete modeling of systems, components, their interfaces, protocols, and protocol behaviors. Several different views of these interfaces are presented. Only selected views need to be used to address specific concerns (e.g., logical data flow, protocol stacks and behavior, message definition, physical interconnection). It is important to understand that while the method has the ability to model interfaces, protocols, and their behavior, end-to-end performance, and other aspects of the interfaces**, it is not required to use all the features of the method for every problem**. Appropriate application of the method, starting with the basics, allows other views to be added when and where they are required. It can be applied in its simplest forms in the earliest stages of system modeling, and elaborated, as needed, when additional features are needed.

II. Systems of Systems Interface Considerations

There are some fundamental assumptions and considerations for systems-of-system modeling, and these can be succinctly stated:

- Each major system has interfaces
- The elements within each system have interfaces
- Interfaces include the connection points on the interacting elements, the data items that are exchanged, the constraints and/or rules that govern the exchange, and the medium for the exchange (i.e. link)
- Multi-layered communication interfaces include application to physical layers
    - For some interfaces the link aspects can drive performance (e.g. atmospheric affects of various signals)
    - For other interfaces protocol behavior and data transformations can drive performance (e.g. end-to-end throughput)
- Different views are used to address different concerns
- Different interface abstractions, from logical flows to complete protocol stacks, are employed as needed
    - End-to-end data flows, connectivity, and data transformations are shown where required
    - Physical connections
    - Protocol specifications
    - Message definitions
- Complete interface specifications that span discipline concerns are supported. The type and detail of interface information for each view must be consistent. The use of the proposed modeling method with SysML directly supports the achievement of this goal.

A. A simple example of an interface specification

Before getting into the details of modeling, it is useful to establish the fundamental features of protocol interfaces. In their simplest form two components may be modeled as boxes and the connection between them as a line, as in Figure 1 which is re-used from our INCOSE IS2015 paper on a modeling pattern for layered system interfaces. [7]

This simple view shows “boxes and lines”, a frequent representation of component interconnections. But what are the interfaces? Where are the protocols? As shown, the two components, Sender and Receiver, exchange PDF formatted files. It does provide a simple End-to-End view of sending a PDF file from A to B, but no details are provided about the interfaces used to perform the transfer. This file transfer is essentially the high level requirement, what the user cares about. For some purposes this sort of representation may provide sufficient information, but it is not a specification that can be used in any rigorous way to support design or implementation.

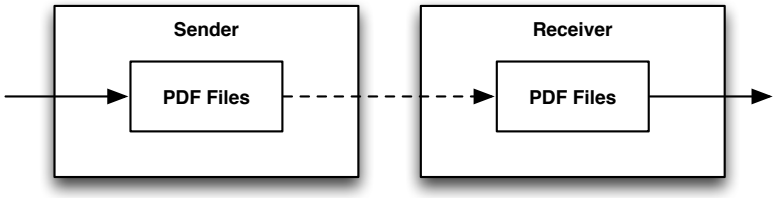

Figure 1. Simple file exchange

B. A simple interface stack specification

In order to provide enough detail to understand the interface between Sender and Receiver it is necessary to describe not just the data that is exchanged, but also the means used to exchange it. This might be the protocol stack which is shown in a simple form in Figure 2.

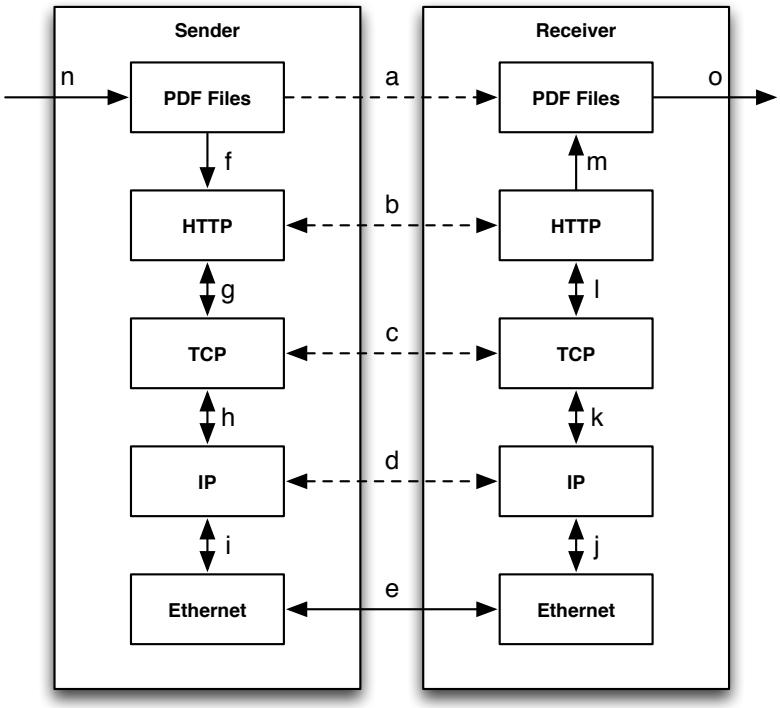

Figure 2. Simple file transfer interface showing protocol stack

While Figure 2 is only a rudimentary interface specification, it does provide visibility into some essential aspects of the Sender - Receiver interface, and of protocol interfaces in general:

1. An interface on a component is defined by a protocol stack. The protocol stack consists of vertical layers, where each layer is a protocol entity.
2. This particular interface is implemented with protocol entities that include HTTP, TCP, IP and Ethernet, but that will not always be the case.
3. Both components use the same stack at either side of the interface.
4. The protocol stacks in each component are connected both horizontally and vertically.
5. The horizontal links (a, b, c …) are logical connections, they exchange Protocol Data Units (PDU) between peer protocol entities.
6. The vertical links (f, g, … l, m) use the provided (upper) and required (lower) interfaces of the protocol entities to exchange Service Data Units (SDU) between layers.
7. The logical flow at each protocol layer is between peer entities in the interfacing components.
8. The actual flow of data is down the stack, across the physical layer interface (e), and then up, the stack.
9. At the physical layer (Ethernet in this example) all of the PDUs of all the layers are visible “on the wire”.

It is possible to use different views to permit separation of concerns. A view can focus on just the TCP layer, Sender to Receiver, across the "c" interface. This view is focused on how entities in the layer are connected (horizontally, between peer protocol entities) and how they behave (horizontally, within and between these two protocol entities) as they exchange PDUs.

Another view could focus just on the stack of protocol entities. Such a view might focus on which protocol entities are selected, i.e. why TCP instead of UDP, how the stack is connected (vertically, where SDUs are exchanged), and how the stack behaves (vertically, which operations are required on the SDUs of an upper layer to meet the provided interface of the lower layer).

Simple lists of interface protocols are not sufficient to understand and specify even this simple file exchange interface. Space data systems, with their usual highly asymmetric links, and intermediate service elements such as ground stations, require additional details in order to be fully specified end-to-end. The next section describes the means to do this.

III. The Interface Modeling Method

The interface modeling method supports the definition and refinement of systems-of-system interfaces at different levels of abstraction and different levels of detail. The following sections describe different views of the protocol interface and how to construct them.

1. End-to-End view of the whole system, in context, showing data flows
2. End-to-End protocol view, showing the most significant protocols supporting the data flows
3. A "black box" view showing one element of the system and its decomposition
4. A "white box" view showing the protocol stack that implements an interface
5. The interfaces of a single protocol entity
6. A state machine showing the internal behavior of a protocol entity
7. Specifying the standards that define interface compliance

Other aspects, such as interface and protocol performance may also be specified in a way that supports direct analysis of the model; for those details see the paper by Sarrel and Simpson.

A. End-to-End View of the Whole System

There are many different ways to model systems. Most methods start with a top level view of the system that provides context. Figure 3 shows an example End-to-End context view for an example space data system. This shows the data flow between the observed data source and the end user, via the Spacecraft and the Ground System. While this figure shows the major physical elements of the system that is being modeled, it is intentionally somewhat abstract and provides only a coarse grained decomposition of these elements. One of the benefits of using an MBSE tool is that it can manage the internal consistency of the model, so we can focus on specific details, or leave them out where needed for clarity.

This figure also shows the nature of the connections between elements. The connectors or links also have additional details and specification, which are hidden in this view to emphasize the context and the end-to-end flows. Similarly, the ports just indicate direction of flows through them. The internal details of the interfaces and protocols are not shown in this view, but they can be captured in the model at the level of abstraction that is required and seen in other views.

None of these figures show organizations, ownership, or operational details, but views that do show such information could be developed as an overlay on this context diagram. A variety of simple embellishments can be used, such as using color to indicate ownership or adding these kinds of properties to the modeled elements themselves. Similarly, if operational details are needed, they can be added in a separate set of views that reference these elements and/or their decomposition and viewed from another perspective.

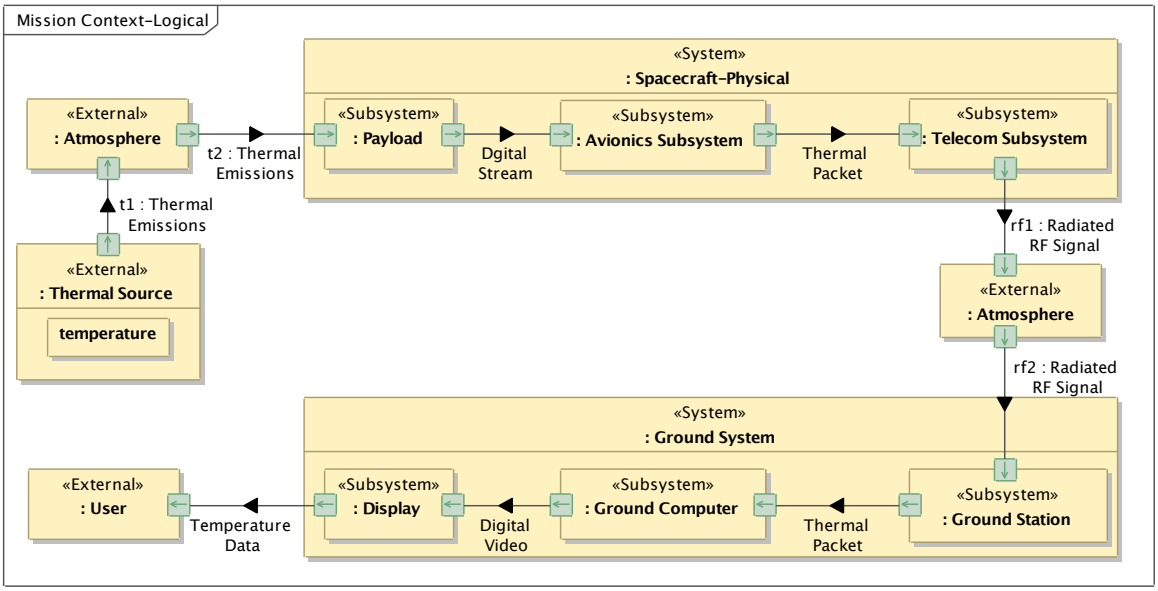

Figure 3. End-to-End contextual view of a space system

Most elements shown in this figure have some sort of stereotypes applied, such as <<Component>> or <<Protocol Entity>>. Good practice for modeling large systems is to develop a profile or a set of stereotypes to represent domain specific definitions that can be used consistently. Figure 4 shows an example that was used for a recent system-of-systems modeling project. Similarly, the stereotypes might include a color code that permits visual distinctions of subsystems, or object types, or ownership, as needed for the particular model.

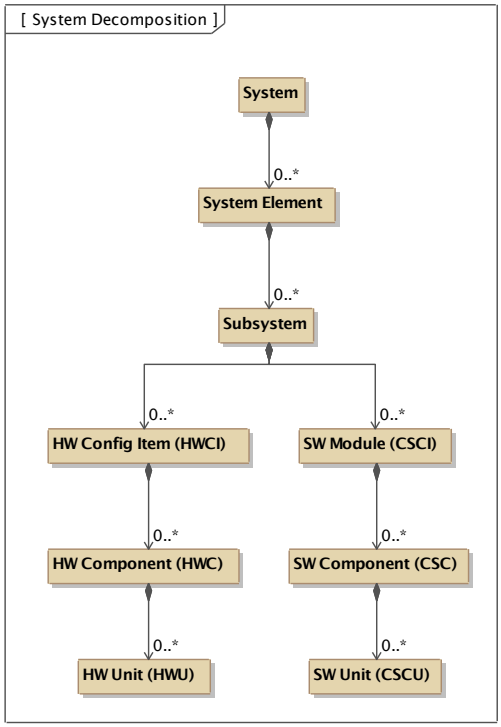

Figure 4. Example system stereotype definitions

This class structure example includes definitions of system, element, and subsystem, as abstract entities. At the levels below subsystem, a distinction is drawn between hardware (HW) and software (SW) items. Keeping the model elements abstract down to some agreed level, and only characterizing implementation distinctions (HW / SW) below that level is a way of approaching initial architectural modeling at an abstract level before drilling down to

implementation details. This also permits added flexibility in mapping from architecture to one or more possible implementation approaches.

B. End-to-End Protocol View

Figure 5 shows the major End-to-End communications systems and also the high level protocol and flows. This figure is a re-statement in SysML of a typical cross support deployment using CCSDS standards. Similar figures in the RASDS style appear in the CCSDS Space Communication Cross Support Architecture Requirements Document, CCSDS 901.1-M-1 [8], but this has the more familiar Spacecraft, Ground Station, and MOS terminology.

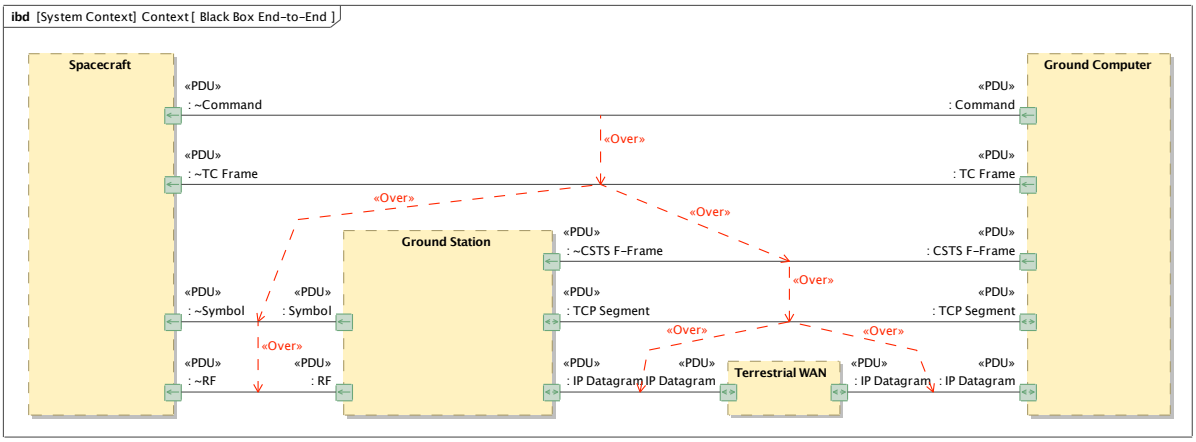

Figure 5. End-to-End contextual protocol view of the whole system

Commands to the spacecraft flow in CCSDS TC frames from the Ground Computer to the Spacecraft but the actual data flows at the layers below this use a disjoint set of protocols. The Ground Computer does not directly radiate the TC frames on the RF link to the Spacecraft, it uses the services of the Ground Station. The Ground Station may be owned by another agency or a commercial service provider. The Ground Computer in this example uses a standard service interface called the Cross Support Transfer Service (CSTS) forward frame (F-Frame). This service accepts CCSDS frames, with whatever data they carry for delivery to the Ground Station. It is the Ground Station that encodes the data and radiates the RF signals. There is an analogous, but different, service interface for the return link, and it provides both on-line and off-line services along with signal and service quality annotations. A compliant Ground Station will also have service interfaces for planning, scheduling, and configuring the link, and other service interfaces for monitor data, radiometric data, and possibly service control. These are not shown here, but they follow the same pattern.

These forward and return data paths are highly asymmetric, and this sort of diagram allows this to be clearly specified. On the terrestrial side of the Ground Station the CSTS service runs over standard Internet protocols and makes use of private or commercial WAN services and routers. On the space link side of the Ground Station CCSDS space link, coding, modulation, and selected RF signals are used to communicate in space. This figure shows the typical CCSDS space communications standards in relationship to major system elements, and shows that the upper layer command and TC frame data has an <<Over> relationship to the underlying, and distinct, protocol stacks.

A diagram like Figure 5 shows the relationships among major elements and some key details of the end-to-end flow, and may form part of an Interface Control Document. At the same time, this figure shows only the top level ports, protocol types, and data flows, and not the details of the interfaces nor the protocol stacks.

C. "Black Box" View Showing Decomposition of One System Element

In order to more fully specify the structure of the system, it is often necessary to define how the system is decomposed. This may be done either top-down or bottom-up; either as a decomposition activity or one of composing systems of subsystems and components. It is often natural to think of decomposing the system, but in reality, systems are often composed by assembling pre-existing components, such as computers, buses, physical data links, receivers, antennas, and, for spacecraft, instruments. Figure 6 shows a view of the Spacecraft element from the End-to-End view.

This figure only shows a partial decomposition and some component relationships, since a typical spacecraft will have many more subsystems and much more complicated interconnections than this simple example showing Avionics and Telecom. This view can be extended to include any number of other systems and elements. This decomposition

also shows hardware and contained software. These components are connected at a port and the data that is exchanged is typed as packet data. The port has directionality, one-way in this example, and the port is shown on all parts that participate in the connection.

This Black Box view is useful in that it shows hardware and software relationships and the presence of key interfaces and connections. As with the rest of the figures to this point, it shows only the top level ports and data flows, but not interface details or protocol stacks. For many purposes, a view with this level of decomposition and connectivity will be sufficient. For other purposes more information must be provided to define the details of the interfaces.

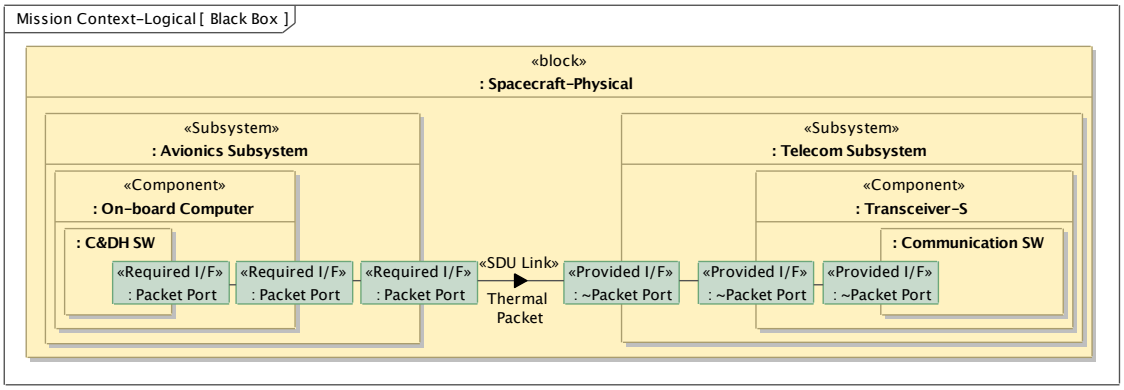

Figure 6. "Black Box" view of the Spacecraft Element

D. "White Box" View Showing Interface Protocol Stack Details

A "White Box" view may be used to show an internal view of the details of the protocol stack that implements the interface. The modeling pattern is to show nested parts within the port. Each part is a protocol entity that is typically identified as belonging to a specific protocol layer, as described in the ISO BRM. When specific, pre-defined, layers are used it is good practice to retain those terms, but other terms may be introduced as needed. For instance, when the BRM was defined there were not the sort of messaging layers in use that are quite common today. Ideally, each of these layers will be associated with a specific standard, as shown in Figure 7.

A specific intent of this view is to specify the complete stack of protocols that define an interface. This starts at the application data transfer layer, and ends at the physical layer. Not all of these layers need to be shown on every diagram, and any given model view may stub off the details when they do not provide any added useful information. A term used for the stack of protocols exposed at the interface of a component is the Interface Binding Signature.

Figure 7 includes the following layers, with these typical characteristics:

- Application layer: packet transfer protocol, manages exchange of packet data between applications.
- Transport layer: Transmission Control Protocol (TCP) [9], provides end-to-end delivery of data, complete, once only, and in order.
- Network layer: Internet Protocol (IP) [10], provides network layer routing over any number of intermediate network nodes.
- Data link layer: 1 Gb Ethernet, provides data link layer services that may involve a fabric of switches and hubs.
- Physical layer: twisted pair cable (Cat-5) and RJ-45 plug terminations.

In this figure, the data flow between the source Packet Processor and the destination Packet to Frame Processor is actually shown twice. The top level connection is a high level view, much like in Figure 6. Below that is shown the details of the protocol stack. Usually, only one or the other is shown in a view. The logical connections at a horizontal layer are labeled as <<PDU Link>>, to show that the data that flows here is actually PDU's of the protocol entities at that layer. The characteristics of these PDU's differs from layer to layer as does the behavior of the protocol entities.

As in Figure 5, the end-to-end flow of data within a subsystem may not use a homogenous lower layer structure. For instance, below the network & transport layers (TCP/IP) a variety of other data link and physical layers may be employed, or intermediate devices such as Ethernet switches may be used. Depending on the purpose of the diagram, these details may be hidden or exposed or addressed in a separate view. This diagram also reflects that the physical layer ports (1GE and RJ45) are hardware and actually belong to the Transceiver and Computer components, while the upper protocol layers are software and belong to the Communications and C&DH SW.

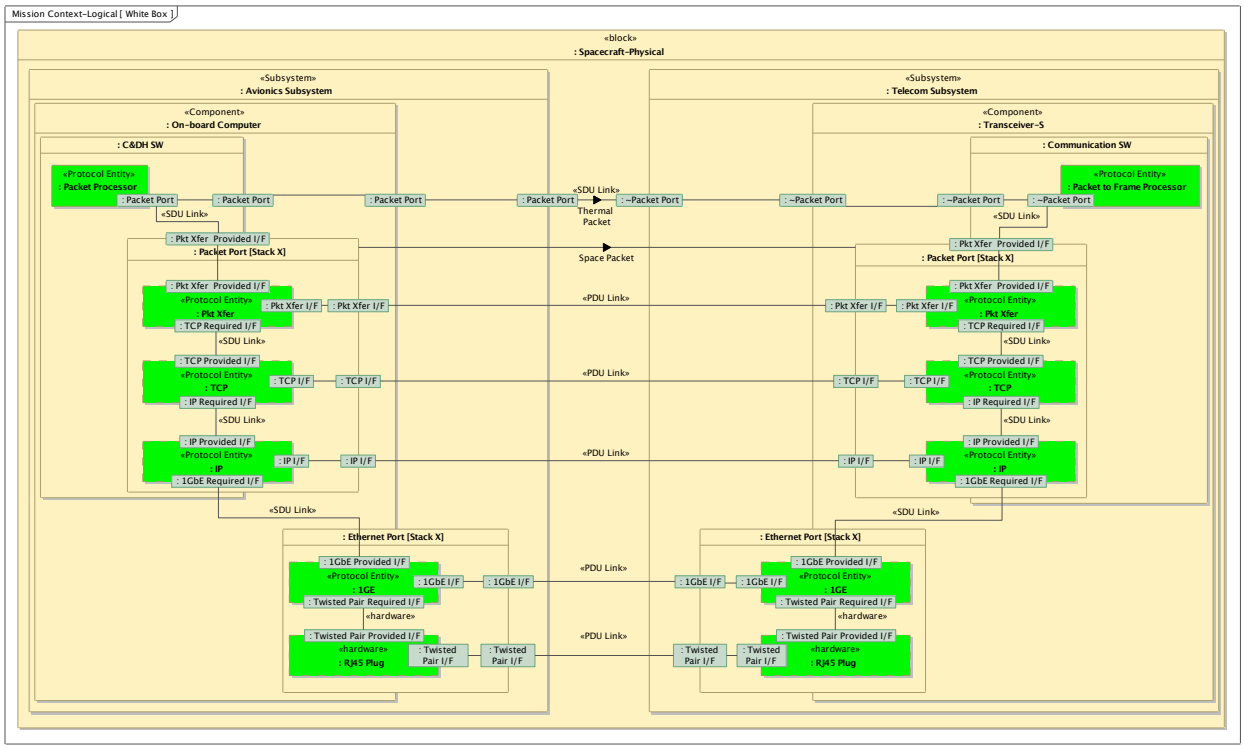

Figure 7. "White Box" view showing the Spacecraft Element protocol stacks

Frequently a reference to "TCP/IP" or Ethernet is used as a short-hand, when in reality each comprises several separate specifications with their own explicit sets of details. A nearly ubiquitous example of this is the USB interface [11]. The "USB spec" for this seemingly simple interface, used on "thumb drives" and other common devices, is actually several quite complex specifications dealing with the physical interface, signaling, cabling characteristics, bus control, device discovery, and device type related behaviors, to say nothing of at least three different backward compatible versions of this evolving spec.

For some purposes it will be sufficient to just identify the interface binding signature, and this will usually be the case when well known protocols with well understood behavior are selected. In other cases it may be necessary to model in detail the interfaces or behavior of a new or specialized protocol.

E. Interfaces of a Typical Protocol Entity

For purposes of modeling protocol entities, each one is considered to have three ports: the interface that provides 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. These are shown in Figure 8. There may also be a management interface that can be used to configure and control the protocol entity, and that may be in-line with normal PDU data flows or via some separate port. In some cases this "management interface" may be provisioned as a Management Information Base (MIB) that is loaded at boot time, or it may even be complied in as selected options that are implemented (or not).

Description of protocol entity ports:

- Provided Service port (TCP this in example): the services offered to any upper layer (N+1) protocol, defined as an abstract service and using a layer N Service Data Unit (passes SDUs)
- Required Service port (IP this in example): the services required from any lower layer (N-1) protocol, defined as an abstract service and using a layer N-1 Service Data Unit (passes SDUs)
- Peer Protocol port (TCP PDU in this example): 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 (exchanges PDUs)
- Protocol management may be in-line with the protocol, via a MIB, or a separate interface

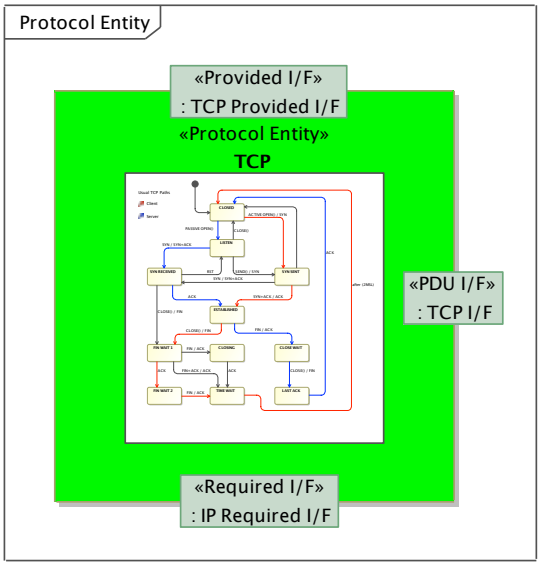

Figure 8. Interfaces of a Protocol Entity

It is expected that the required services from the N layer and the provided services from the N-1 layer will match, but sometimes these are developed at different times, or new technology gets introduced after the fact. In the case of IP, which is used in some of these examples, several generations of underlying data links have come and gone since that protocol was first designed in 1981. It is really an implementation detail and not often modeled, but there is actually a shim layer between the IP implementation and the possible underlying data links. These shims ensure that the IP datagram, the PDU specified by RFC 791, is broken into parts and then re-assembled, as needed, to fit into the underlying data link layers. There is a whole suite of "IP over xyz" RFCs that describe in detail how this is done. This can be thought of as "impedance matching" between TCP and the data links. In CCSDS, the Space Packet Protocol [12], and the Encapsulation Packet [13], provide similar functions.

F. State Machines Define Protocol Entity Behavior

At the peer entity (PDU) port, each protocol entity exposes its defined behavior. Figure 8 is a Black Box view of such an entity, but the "badge" on the front gives a hint of the complexity that lies within. The badge, by the way, is just incidental decoration, it may be used if it is found useful, but serves no modeling purpose aside from decoration.

The description of actual protocol behavior may be specified in a number of different ways in standards. All of the following (and more) have been used:

- State machines (sometimes shown as an ASCII diagram)
- State tables
- Sequence diagrams
- Simple English text

For reasons of clarity and ease of understanding, state machines are recommended as clear, formalized representations that may be well integrated with the rest of the model. In typical protocols, the behavior at a given layer is actually performed by a pair of cooperating state machines. To capture these dynamic exchanges, the state machine diagrams may be accompanied by sequence diagrams that show temporal exchanges between the peers.

State machines can be used to describe the process that a protocol entity follows when it receives a protocol Protocol Data Unit (PDU) from a peer entity. They can be used to describe the exchange(s) of PDUs between peers, and they may be used to describe the behavior at the required and provided interfaces, such as start-up, connection establishment, and SDU transformation into PDUs. The behavior may involve the dynamics of PDU exchanges, including both nominal and error conditions.

Figure 9 is a part of the state machine that is at the heart of the TCP protocol, as defined in RFC 793 [9]. A crude, but effective, variant of this diagram is present in that document in the form of an "ASCII drawing". TCP is a connection oriented protocol and this figure shows the behavior of both peer entities, the Client (red lines) and the Server (blue lines), but it only shows the connection establishment and tear-down processes.

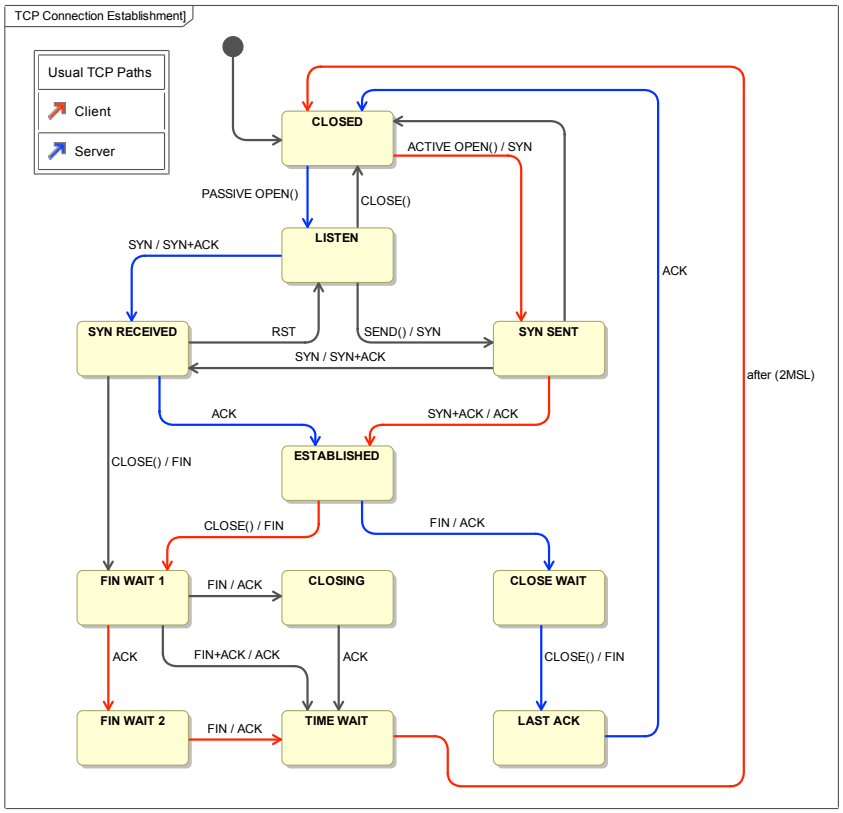

Figure 9. TCP example of a State Machine - Protocol entity behavior

The bulk of the work that TCP does to ensure reliable, once-only, in order, delivery of data without omissions is done inside the Established state shown in the middle of the diagram, and this behavior is considerably more complex.

Most of the TCP specification is done in the "simple English text" style, and this is eighty-five (85) pages of rather dense "prose". A paper on finite state machine modeling of the TCP standard [14] provides a translation of the TCP spec into an extended finite state machine (EFSM) model using EFSM/SDL. This state machine model is extensive, at thirty-seven (37) pages of diagrams, but it is quite accessible and understandable.

Furthermore, simulation of the behavior of these state machines can be performed and the behavior verified, see the paper on EFSM modeling for an example. A clear state machine description is valuable for understanding existing protocols, and it should prove especially useful when designing new protocols.

G. Interface Compliance Specification

Although the concept was not introduced earlier, in protocol stack views such as Figure 7, it is possible to directly specify in the model which interfaces in the full stack are defined by, and comply with, some standard specification. It is also possible to tie the details of each layer to a whole standard or to specific sections of a standard if that is required. This concept of modeling compliance with a standard is captured explicitly in the model by the "Satisfies" notation shown in Figure 10.

Compliance statements may include interface data, PDU structures, behavior (preferably as described by one or more state machines), or by activity, and sequence diagrams. If the protocol at any given layer is fully specified in a published standard, just using the "Satisfies" relationship for these layers may be adequate. In such cases the specification of the interface binding signature may be accomplished by describing the set of protocols used at the interface in a black box diagram, identifying the "satisfies" relationships, and then just referencing that interface specification wherever needed. These components are indicated by dashed lines as shown in Figure 10, indicating that the parts are reference parts that are not necessarily owned by the enclosing block.

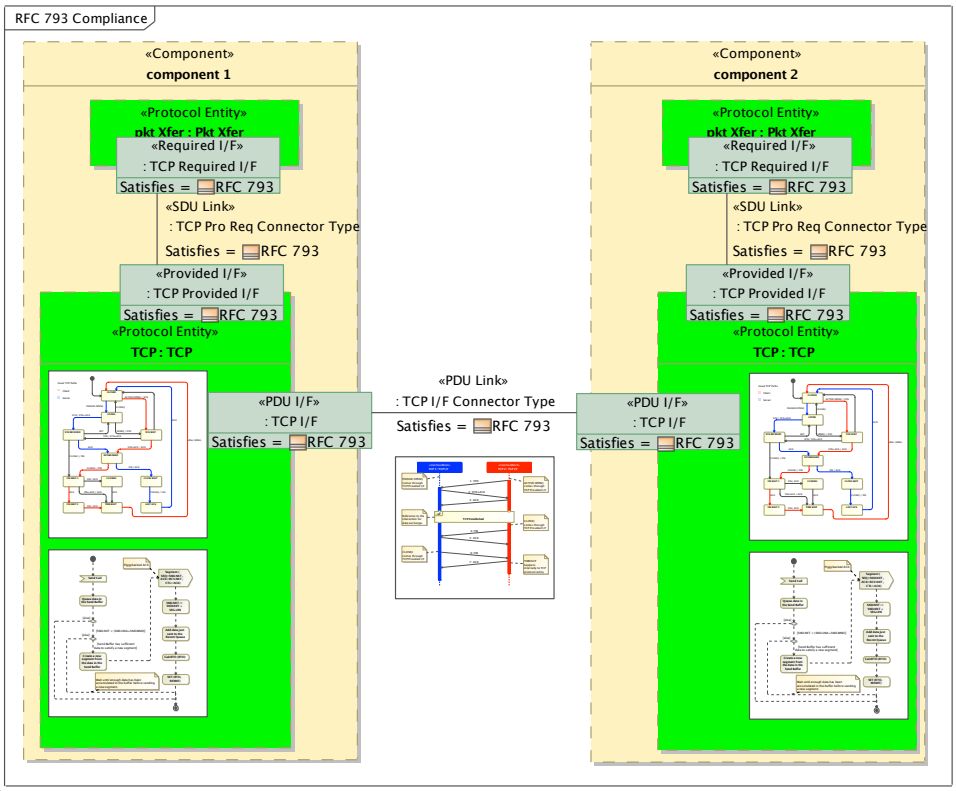

Figure 10. Interface Compliance Specification, TCP Example (RFC 793)

Figure 10 also shows the relationships that exist between the protocol specification, the state machines that implement the behavior, and a sequence diagram that describes the time-ordered PDU exchanges between the two entities. The sequence diagram shown is a hyperlink to the actual diagram in the model.

IV. Conclusion

This paper describes a method for modeling space data systems and element interfaces and behaviors at successive levels of detail. This supports both abstract views and deep specification of technical interfaces and behavior, where these details are required. Careful use of the method from the outset, will permit further elaboration of details as needed.

A. Modeling Flexibility – Consistency and Re-use of Components and Views

Models may contain multiple views, from top level context, to composition and components, down to low level interface and protocol details. Careful selection of views allows different levels of abstraction to be described, and supports drawing clear distinctions between architectural level views and implementation level details.

Creating defined stereotypes, and libraries of components and protocol entities, allows them to be re-used and combined as needed in different views. Whether developed top-down or bottom-up, these libraries may be used as building blocks for new models, promoting consistency and clarity. Libraries of components and protocols may find re-use within a particular system application and across projects.

The different views allow the systems and systems-of-systems to be understood from different perspectives and at different levels of detail. SysML does includes support for viewpoint specification and view construction, but this requires some discipline from the modeling team. SysML, and the typical modeling tools help ensure consistency across the model.

B. Systems-of-Systems Interface Modeling Benefits

The method presented here permits space data systems, and systems-of-systems, their structure and interfaces, to be modeled with a high degree of fidelity. Using a principled approach to system, component, interface, and protocol modeling allows successive levels of detail to be added as and when needed. Even a partial model can document

systems, interfaces, and behavior at a much deeper level of specificity than is provided by an ICD "protocol list" or a set of "boxes and lines" diagrams, whether produced by graphical presentation or modeling tools.

Particularly for space data systems, which may include elements from different organization, distinctive and asymmetric protocols, and other interoperability complexities, adoption of a clear and consistent approach to modeling the interfaces can offer significant benefits. Including clear descriptions of connectivity and composition, and mapping to specifications and behavior can help provide unambiguous specification of system structures and component interactions. The development of these models can also support analysis of flow and continuity, and verification of interface behavior, and support analysis of end-to-end connectivity, behavior and performance.

Acknowledgments

Aside from the many hours that the co-authors spent discussing and refining these concepts, discussions with many other people helped to refine this method. This modeling pattern was first developed and applied as part of the Space Communication and Navigation (SCaN) Integrated Network Interface Definition Trade Studies [15], and the related SCaN Network Integration Project (SNIP) [16].

There has been other work to model interfaces in SysML, some of which modelled certain aspects of similar layered interface concepts. A layered interface pattern was applied by Maddalena Jackson to describe data flows in support of human space flight Ground Data System of the Exploration Flight Test 1 (EFT-1) project [17]. This included the flow of information across the ground network supporting the mission and used two layers. The top layer showed the flow from source to destination and the second layer described the connections between the routers, switches, firewalls and servers. Constraints were added to describe the path so that connectivity could be evaluated directly from the model. Robert Karban developed and applied interface and protocol stack patterns to model software, electrical, optical, and mechanical interfaces while at the European Southern Observatory [18]. Takahiro Yamada, Erik Barkley, John Pietras, and others contributed substantially to the development of the CCSDS RASDS and SCCS-ADD which defined the view viewpoint methods and then tested them on a major document.

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] Maier, Mark, *Architecting Principles for Systems-of-Systems,* Systems Engineering 1, John Wiley & Sons, Inc., 1998, pp 267–284.

[2] *OMG Systems Modeling Language (OMG SysML™)*, Version 1.4, OMG Document formal/2015-06-03, Object Management Group, September 2015. [2]

[3] *Information Technology—Open Systems Interconnection—Basic Reference Model: The Basic Model*, Issue-1, International Standard, ISO/IEC 7498-1:1994, Geneva: ISO, 1994.

[4] *Systems and software engineering — Recommended practice for architectural description of software-intensive systems*, ISO/IEC 42010, International Standards Organization, July 2007, revised 2011.

[5] *Reference Architecture for Space Data Systems*, Issue 1, Recommendation for Space Data System Practices (Magenta Book), CCSDS 311.0-M-1, Washington, D.C., CCSDS, September 2008.

[6] Shames, Peter M, Sarrel, Marc A, Freidenthal, Sanford A, *A Representative Application of a Layered Interface Modeling Pattern*, 26th Annual INCOSE International Symposium (IS 2016), Edinburgh, Scotland, UK, July 2016 (submitted for publication).

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

[8] *Space Communications Cross Support—Architecture Requirements Document*, Issue 1, Recommendation for Space Data System Practices (Magenta Book), CCSDS 901.1-M-1, Washington, D.C., CCSDS, May 2015.

[9] Postel, J, *Transmission Control Protocol,* STD 7, Reston, Virginia, ISOC, September 1981.

[10] Postel, J, *Internet Protocol,* STD 5, Reston, Virginia, ISOC, September 1981.

[11] *Universal Serial Bus Revision 2.0 Specification*, USB Version 2.0 (with updates), Oregon, USB Implementers Forum, Inc (USB-IF), April 2000.

[12] *Space Packet Protocol,* Issue 1, Recommendation for Space Data System Standards (Blue Book), CCSDS 133.0-B-1, including Technical Corrigendum 1 and Technical Corrigendum 2, Washington, D.C., CCSDS, September 2012.

[13] *Encapsulation Service*. Issue 2. Recommendation for Space Data System Standards (Blue Book), CCSDS 133.1-B-2. Washington, D.C., CCSDS, October 2009.

[14] 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, Kent State University, 2005

[15] Shames, Peter, Anderson, Michael, et al, *NASA Integrated Network Monitor and Control Software Architecture,* SpaceOps 2012, AIAA, June 2012.

[16] Uzo-Okoro, Ezinne, Hudiburg, John, et al, "*NASA Space Communication and Navigation Network Integration Project (SNIP),* AIAA 2014-1653, SpaceOps 2014, AIAA, May 2014.

[17] Jackson, M, et al., *Architecting the Human Space Flight Program with Systems Modeling Language (SysML)*, AIAA 2012-2556, Infotech 2012, AIAA, June 2012.

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

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). Within 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 Documents (SCCS-ADD, CCSDS 901.0-G-1 and SCCS-ARD, CCSDS 901.1-M-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.'