25th Annual INCOSE International Symposium (IS2015)
Seattle, WA, July 13 – 16, 2015

A modeling pattern for layered system interfaces

Peter M. Shames
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91009

Marc A. Sarrel
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91009

Abstract. Communications between systems is often initially represented at a single, high level of abstraction, a link between components. During design evolution it is usually necessary to elaborate the interface model, defining it from several different, related viewpoints and levels of abstraction. This paper presents a pattern to model such multi-layered interface architectures simply and efficiently, in a way that supports expression of technical complexity, interfaces and behavior, and analysis of complexity. Each viewpoint and layer of abstraction has its own properties and behaviors. System elements are logically connected both horizontally along the communication path, and vertically across the different layers of protocols. The performance of upper layers depends on the performance of lower layers, yet the implementation of lower layers is intentionally opaque to upper layers. Upper layers are hidden from lower layers except as sources and sinks of data. The system elements may not be linked directly at each horizontal layer but only via a communication path, and end-to-end communications may depend on intermediate components that are hidden from them, but may need to be shown in certain views and analyzed for certain purposes.

This architectural model pattern uses methods described in ISO 42010 [1], Recommended Practice for Architectural Description of Software-intensive Systems and CCSDS 311.0-M-1 [2], Reference Architecture for Space Data Systems (RASDS). A set of useful viewpoints and views are presented, along with the associated modeling representations, stakeholders and concerns. These viewpoints, views, and concerns then inform the modeling pattern. This pattern permits viewing the system from several different perspectives and at different layers of abstraction. An external viewpoint treats the systems of interest as black boxes and focuses on the applications view, another view exposes the details of the connections and other components between the black boxes. An internal view focuses on the implementation within the systems of interest, either showing external interface bindings and specific standards that define the communication stack profile or at the level of internal behavior. Orthogonally, a horizontal view isolates a single layer and a vertical viewpoint shows all layers at a single interface point between the systems of interest. Each of these views can in turn be described from both behavioral and structural viewpoints.

Introduction

Specifying and managing interfaces is at the heart of systems engineering. Well defined interfaces and interoperable protocols allow a designer the freedom to implement the internals of their system independently of the other systems to which it may be connected, and to make

their systems more easily re-usable in different contexts. But simple interfaces from one point of view may actually involve multiple levels of complexity. For example, a USB "thumb drive" is a ubiquitous and simple appearing device, but the document for the USB 3.1 specification alone consists of a primary document and several supplements totaling over one thousand pages that cover physical, logical, service and behavioral specifications.

With increasing use of model-based systems engineering (MBSE) as a tool to help manage complexity in systems engineering, there is a need for a focused method to model interface complexity. The most common language for MBSE is the Systems Modeling Language (SysML) [3], which is based on the Unified Modeling Language (UML) [4]. For the system engineer an MBSE tool combined with modeling patterns and methodology will help form model information into a "single source of truth", support static and dynamic analysis of the models, and interface with external modeling and analysis tools.

This paper is focused on a pattern for modeling layered interfaces, and making this pattern available for reuse by other systems engineers. It is common in systems engineering to represent a single item at different levels of abstraction. In its simplest form, an item may be represented both logically and physically, but in different views. An archetype for such multi-layer abstraction of interfaces is the pattern for describing protocol stacks for computer networks, such as TCP/IP, using the ISO / OSI Basic Reference Model (OSI BRM) [5]. This concept has been adopted and generalized and applied to systems engineering to allow multiple simultaneous levels of abstraction, or layers.

We have defined an interface modeling pattern using SysML that provides a method for modeling interfaces at different levels of abstraction and detail. This pattern is intended as a framework within which users may incorporate as much or as little detail as is needed for their problem domain. That detail may include both structural and behavioral aspects, and is intended to manage the complexity and give a representation of the entire interface. The benefit of this approach is to enable an integrated description and analysis of system interfaces, either statically, dynamically, or both.

This paper presents some key viewpoints, views, and related concerns, leveraging ISO 42010 and RASDS specification of architecture description, and shows how they may be applied to interfaces modeled with our pattern. Describing interfaces is standard practice in systems engineering, but in the case of MBSE a clearly defined modeling practices has been shown to be of benefit. We will present some example applications of the pattern. And, finally, we will outline some future work that could be done to adapt and apply this interface modeling pattern to other engineering domains.

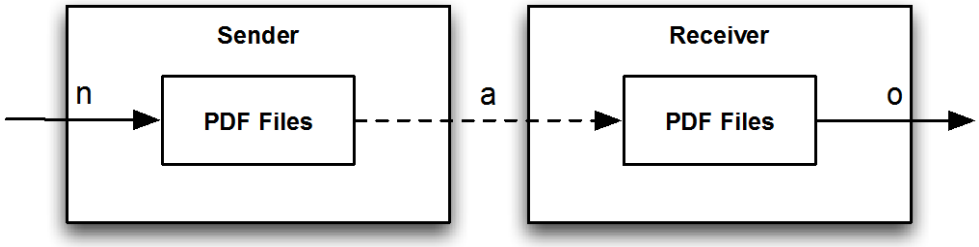

Figure 1. Transferring a PDF file from Sender to Receiver.

Basic Concepts

A basic, familiar, example of two interacting systems is shown in Fig. 1. This figure shows a PDF file being transferred from a sender, perhaps the author of a paper, to a receiver, perhaps

the editor of a technical conference. This first view shows two elements, the sender and receiver, which are transferring a file. It says nothing about the manner in which the file is exchanged. There must be some interfaces and underlying protocols, but these are abstracted away in this applications view. Providing the means for describing the mechanism of the transfer is the focus of the rest of this paper. The OSI BRM standardizes the concepts of a

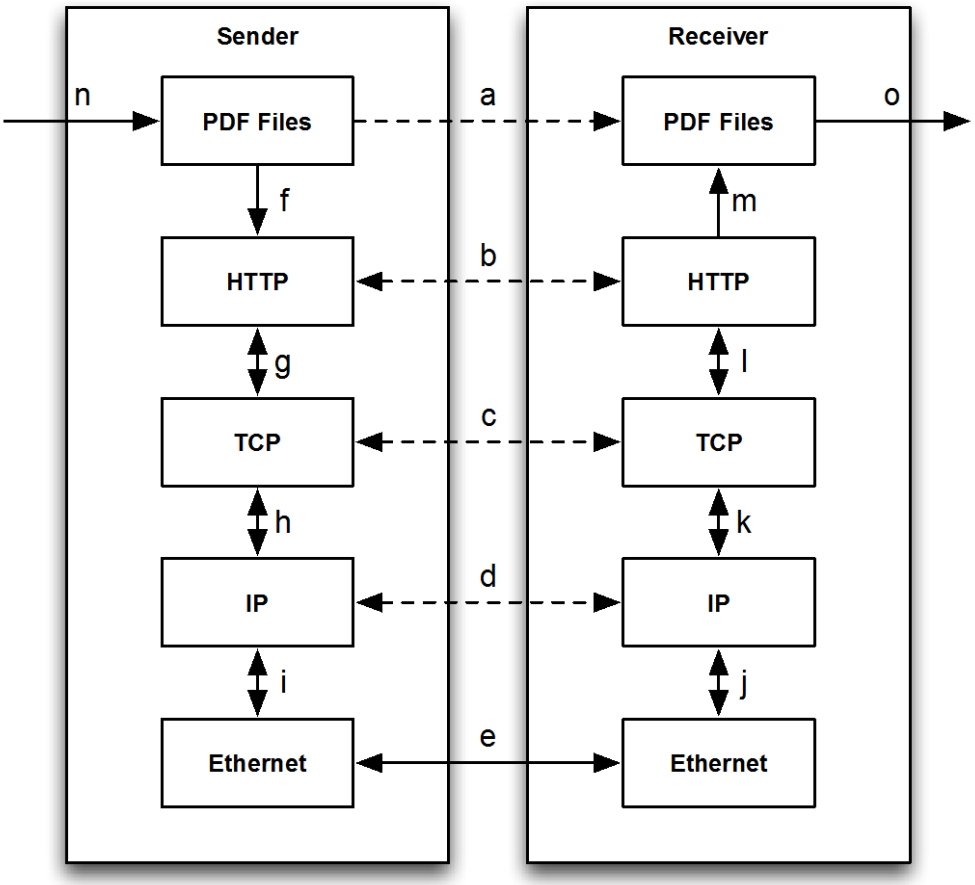

Figure 2. Layered communication beneath the file transfer.

layered communications architecture. This forms the basis for the way that we construct our layered systems interfaces. All the figures in this section comprise a single, unified example intended to be simple and familiar. Each is a different view of the same example at different levels of abstraction. The correspondence between diagrams can be made by matching the names inside the boxes and on the lines.

A more complete model of the exchange is shown in Fig. 2 which exposes four layers of protocol below the exchange of the PDF file itself. Each of the layers has flows in the horizontal direction between peer protocol entities and flows in the vertical direction between adjacent layers. The sender and receiver each have a set of entities that implement the functionality of each layer. At each of these connections and entities it may be necessary to specify interface bindings, controlling standards, requirements, constraints and behavior, and all must be satisfied for the end-to-end communications to perform as desired.

The horizontal connections between peer protocol entities is what determines the interoperability of the sender and receiver. The vertical connections do not govern the interoperability, but must support the behavior of the layers above within each system. Provided interfaces in a lower layer must meet required functionality by an upper layer. Upward transformations in the receiver must match downward transformations in the sender, and vice versa.

The horizontal dashed connections represent logical exchanges between peer protocol entities, the sender and the receiver. The data types exchanged at each horizontal level are specialized and the general term for them is Protocol Data Units (PDU). In this case, the data type across connection 'a' is the PDF file itself. Along connection 'b', it is requests and

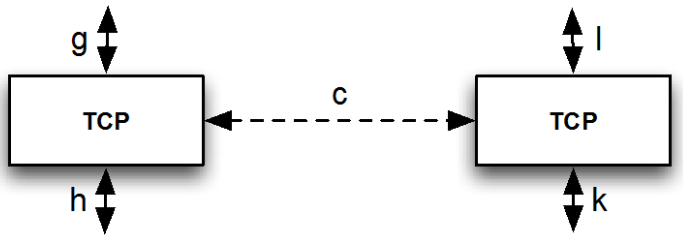

Figure 3. A single layer between the Sender and Receiver.

responses between an http client and server. The bottom three connections, 'c', 'd' and 'e', use TCP sequences, IP datagrams and Ethernet frames respectively. Horizontal exchanges are generally considered to be logical except at the lowest physical layer. Actual data flows go up and down the stack where the lower layers encapsulate PDUs from upper layers. Not all systems need to be modeled down to the physical layer, in some cases the stack may be "stubbed off" at a higher layer.

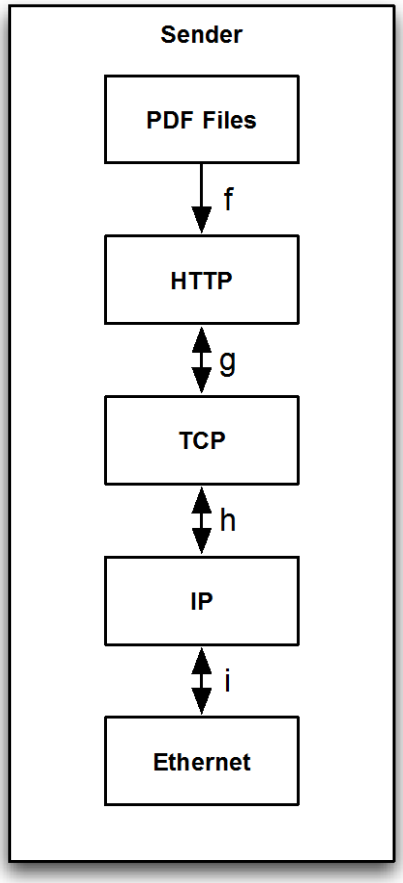

Figure 4. The layers within a single system, the Sender.

Vertical connections represent transformations within a system. It should be emphasized that the data types exchanged vertically between layers are not the same as the data types exchanged horizontally. Conceptually, the data moved horizontally and vertically is the same. But, the packaging, representation and related behavior will be different in practice. The general term for vertical exchanges is Service Data Unit (SDU). Since these vertical exchanges are unrelated to the interoperability of the systems, and are often only described as abstract service interfaces, they need not be the same in the sender and receiver.

At each layer it may be necessary to describe the behavior that occurs across each horizontal connection. The behavior on connection 'a' is that a file is transferred from sender to receiver. Connection 'b' uses HTTP which is a client and server protocol. The client sends requests and the server provides responses. Connection 'c' uses TCP, which is responsible for the end-to-end delivery and retransmission of pieces of the file. This layer transports data end-to-end once, in order and with no omissions. Connection 'd' is IP, which handles point-to-point routing – potentially many hops – of pieces of the data. And finally, connection 'e' is the Ethernet layer, is responsible for logical and electrical signaling and physical cabling between the sender and receiver.

For some purposes it is necessary to describe properties of the connections at each layer. We can specify the speed, in bits per second, of the connection. At higher levels, we can specify, expected rates of error, retransmission, disconnections etc. These properties or constraints may be the basis for static analysis of the interface.

The OSI BRM assigns unique functionality to a total of seven different layers. Our modeling pattern is capable of representing these layers, but nothing in it requires the use of the OSI specification. System engineers can and should define the necessary protocol architectures for their own domains which may contain fewer layers, or more of them. The benefit is that it is possible to separate concerns, and specify properties and behaviors at their appropriate level of abstraction for as many layers as are needed to understand the emergent properties of the architecture.

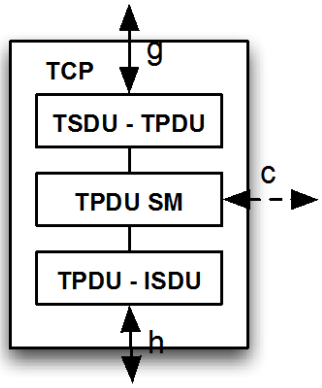

Figure 5. Internal structure of the TCP protocol entity.

In order to address different concerns and focus on different aspects of the systems and their interfaces, it may be necessary to focus on horizontal or vertical slices of the system. Fig. 3 shows a horizontal slice between sender and receiver. This sort of view might be used to focus on interactions at a single protocol layer, especially if many systems are part of the end-to-end chain. Some systems may be disconnected entirely at higher layers. Some lower layers may have complexity and heterogeneity of their own. These views are useful to find possible paths through a large network, and analyze interconnectedness.

In some cases it is necessary to define a vertical slice through a single system, as shown in Fig. 4. This would be of use to someone designing, integrating and testing the internal structure of that system. While this diagram shows just a single vertical path, in reality, much more complex structures are possible. In some cases a single system element may have to bridge two different protocol stacks, where the source stack and destination stack differ significantly.

At each layer in the stack the behavior of the peer protocol entities is usually specified in a standards document, in the form of a state machine or some similar mechanism. The state machines at each end of a connection must interact with each other according to the specification as if they communicated directly with each other via the PDUs at that level of abstraction. Fig. 5 shows a decomposition of the TCP protocol entity. The TCP PDU state machine, TPDU SM, is in the middle. It communicates virtually with its peer in the receiver across connection ‘c’. Above and below the TPDU SM are SDU - PDU adapters. These convert PDUs to SDUs and vice versa. So, a TCP SDU coming from above across provided interface ‘g’ is converted to a TPDU by the TSDU - TPDU adapter and sent to the TPDU SM. If it represents a valid transition of the state machine, it is passed downward, converted to an IP SDU by the TPDU - ISDU adapter and sent out required interface ‘h’. In this way, the two TPDU SMs communicate across the virtual connector ‘c’. The process is reversed when IP SDUs come in required interface ‘h’ from below. Typically, any protocol standards document defines the abstract provided interface and SDUs that must be used to communicate with the layer above. And, each entity may have several choices of PDU - SDU, or abstract service interfaces, to communicate with entities below it.

System Interface Modeling Pattern

Views and Viewpoints

A tenet of system architecture practice is to identify appropriate views and viewpoints to address the concerns of stakeholders, and to identify appropriate means for representing those views. This section outlines some concerns that we have identified and the associated view, see Table 1. Each of the viewpoints and views described below addresses one or more of these concerns. Views may be internal vs. external, horizontal vs. vertical, abstracted or detailed, as required. Any single view may focus on one, some, or all of the systems involved. Individual

Table 1. List of Concerns.

	Concern	View
1	What is the end-to-end construction of the system in terms of major elements?	End-to-End black box view
2	What is the specific stack of protocols needed in each element?	Protocol stack view
3	What is the behavior within a given protocol layer?	Protocol state machine view
4	What are the standards or specification that govern the behavior of each layer?	Interface binding view
5	How are the protocol stacks deployed, end-to-end in order to meet the system requirements?	End-to-End white box view

views may focus only on a pair of elements or on all of the elements in a communications chain. Internal details may be abstracted away or they may be described in depth, as needed.

Behavior, connectivity, and performance may be carefully described such that it may be analyzed or it may be left abstract. The basic pattern defines the elements, interfaces and composition, interface types & binding specifications, and provides the "hooks" for adding further specifications and details. The construction pattern also permits details in any given view to be abstracted away if they are not needed to address a particular concern, but manages all of these different views in a consistent way within the model framework.

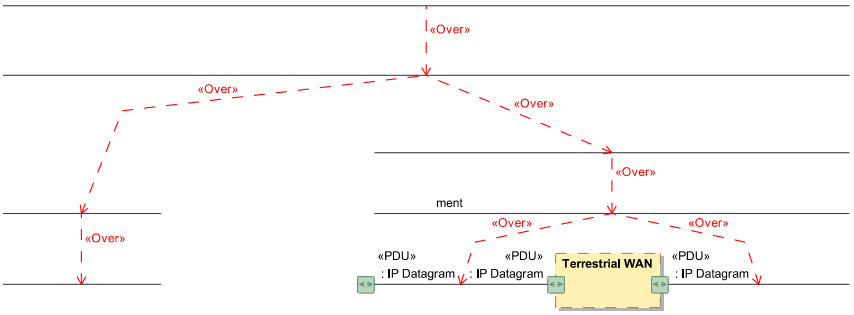

Figure 6. Black box end-to-end view.

The Pattern as Expressed in UML and SysML

These specific examples concern the communications between a Space User Node (spacecraft) in flight and an Earth User Node (mission operations center) on the ground. They are drawn from the CCSDS Space Communications Cross Support Architecture document (SCCS-ADD) [6]. These views show how commands are sent from the Earth User Node to the Space User Node. In between are two intermediary systems, a Earth-Space Link Terminal (ESLT, or ground station) and a terrestrial wide-area network. Space communications links are often challenging because they contain inherent asymmetries in end-to-end protocols. Table 1 lists some of the early concerns that we have identified.

The first view in Fig. 6 an end-to-end black box view, showing components, connections and interfaces, without any internals. This black box view shows all systems and provides an overall perspective of the system. The Over relations show how upper layers are encapsulated

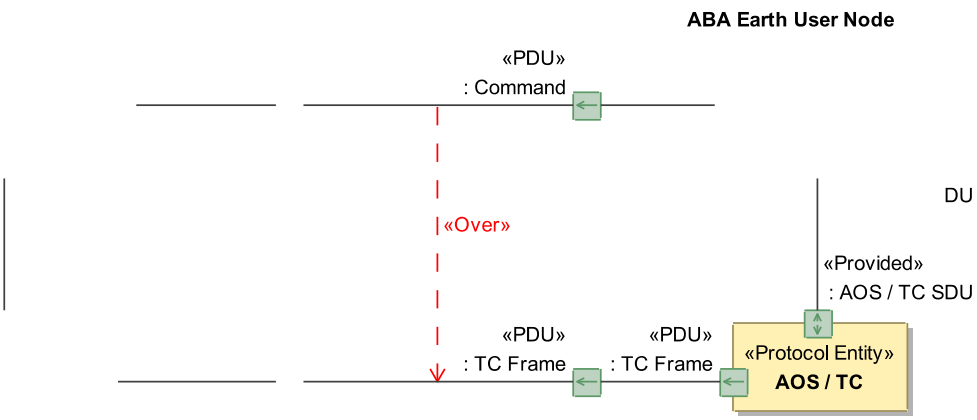

Figure 7. White box user-only view.

into lower layers. The ESLT clearly must do some sort of protocol bridging since its input and output interfaces are different in number and type. The Terrestrial WAN is completely hidden from the Space User Node, yet the latter depends on the former to communicate with the Earth User Node. It may be the case that the Terrestrial WAN is a commercial service, not developed or under the control of the project. In such cases, this modeling pattern can be used to identify and characterize the reliance on such hidden and external systems. All the SysML diagrams in this section comprise a single example. They are all taken from a single, unified SysML model, focused on space data systems. Each is a different view of example at different levels of abstraction. The correspondence between diagrams can be made by matching the names, types and stereotypes inside the boxes and on the lines.

The next view, Fig. 7 isolates the communication between the earth user node and the space user node and abstracts away the intermediate systems and the details of the protocol stacks. This view is useful to describe how the space user node and earth user node communicate commands. Of course, the performance of the link would vary widely depending physical implementation, including the bandwidth and latency of terrestrial connections and of the space link. This abstract view may be relevant whether the spacecraft is in flight, or when the spacecraft is on the ground during assembly, test and launch operations. In reality there would be different protocols and intermediate systems at the lower layers, but this end-to-end view would still be true. This example also illustrates the concept that the Over relations outside the systems must be consistent with the way in which protocol entities are connected inside the systems.

Fig. 8 is a white box protocol stack view that shows the earth-space link terminal and the protocol stacks implemented within it. It shows the details of the bridge function, the F-Frame Production application, within the ESLT that was mentioned in Fig 6. This application bridges the two protocol stacks on the earth- and space-facing sides of the terminal. It translates between the top-most SDUs of the two. The earth- and space-facing interfaces are intentionally modeled to different levels of detail. On the earth-facing side, we chose only to model down to the level of IP, there was little to be gained by modeling further since it is a well established service. An Internet service provider might wish to model this more deeply, but it

is not needed for this purpose. On the space-facing side there are a number of specific protocol choices to be made so we chose to model all the way to the physical layer. The C & S layer manages coding and synchronization, and the RF & Mod layer produces the physical signal that is modulated and transmitted at some RF frequency. In a further refinement of the model all of the specific details of link layer protocol, coding, modulation, and forward / return

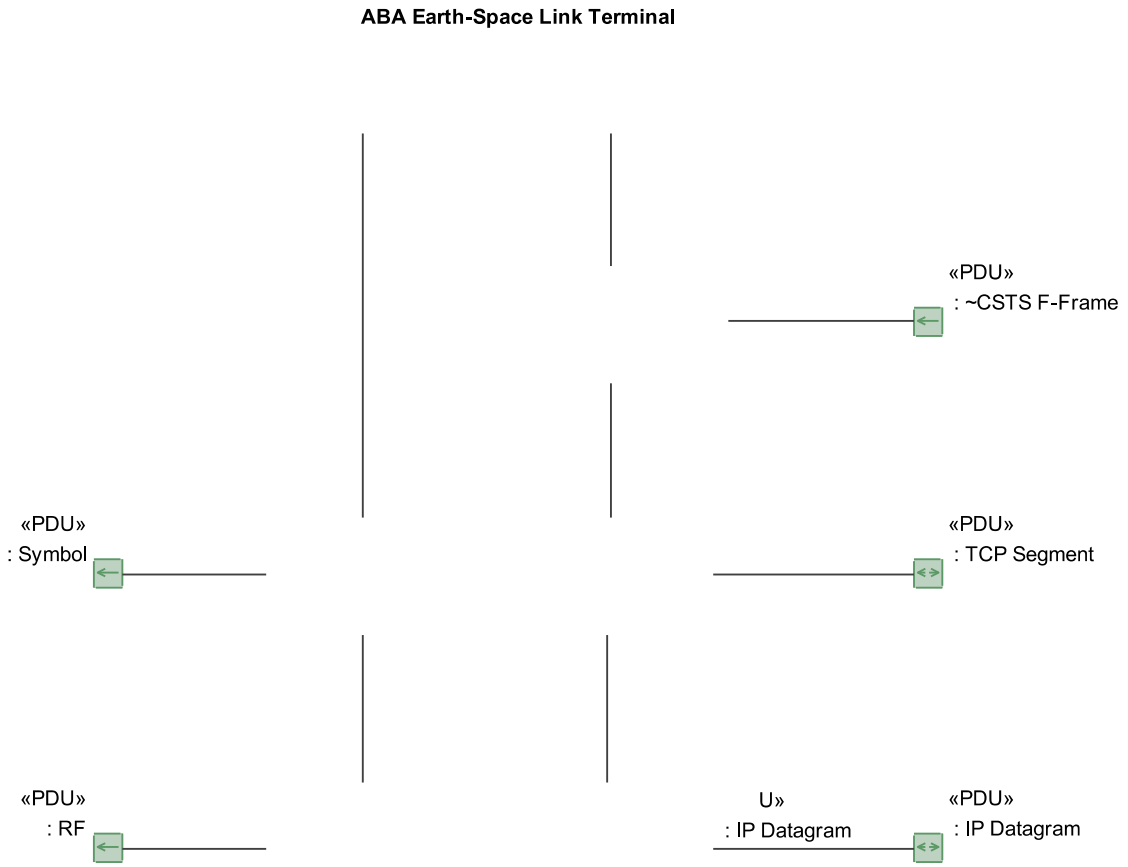

Figure 8. White box bridge for Earth-Space Link Terminal.

frequency may be added to these blocks. In space communications, the uplink and downlink RF links are often very different, in frequency, data rates, and protocols. This view only shows the uplink path, the downlink path would be modeled with different components.

Our final example, Fig. 9, drills into a pair of peer protocol entities in a single layer to show how the logical connection between peers, in this case TCP, is realized using the vertical provided and required interfaces. In this example, the ESLT is the sender and the user node is the receiver. A similar pattern could be used to describe the protocol behavior details in the application bridge shown in Fig. 8. User data is presented to the TCP protocol entity at the upper interface. The sender 1 adapter converts the user data into TCP PDUs. Those PDUs are sent to the TCP state machine. If they represent a valid transition in the state machine they are processed and the resulting TCP SDU is passed down to the receiver 1 adapter where they are converted into user data for the next lower level, in this case IP. The reverse happens on the way up in the user node. But from a logical point of view the PDUs are exchanged directly

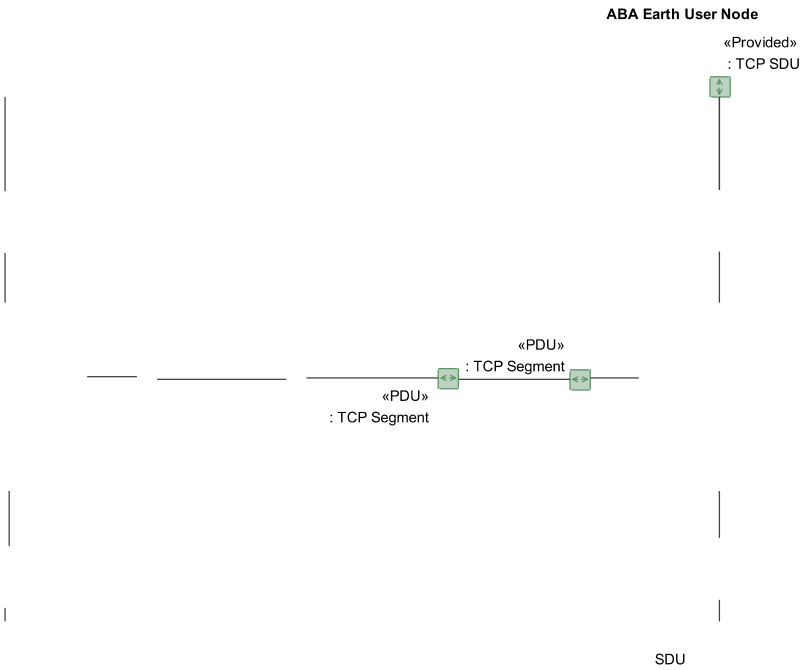

Figure 9. TCP internal state machine.

between the two TCP state machines. In this way, the two state machines remain in sync according to the TCP specification. It should be emphasized again that all these views were produced from a single, integrated model. A change made in one view is automatically reflected in all the others, and all views are guaranteed to be consistent with one another.

Application of System Interface Models

The primary value of this modeling pattern is to allow complex system and interfaces interactions to be decomposed and represented in a self-consistent way. It describes a appropriate set of views and viewpoints that can be used to describe system element connections, behaviors, and constraints at successively deeper level of detail. This pattern, and some similar patterns, have been applied to several real-world problems already.

On the NASA Exploration Flight Test 1 (EFT-1) project [7], we have modeled the ground system of computer networks, using only two levels of abstraction. The lower level of abstraction represents network connections between computers, locations and flight hardware in the system. The higher level of represents required data flows, needlines, from source to destination in a single hop. The primary aim was to show the path along the network that each needline follows. But, it is impractical to specify and maintain an Over relation for each step as the model changes and evolves. There are several hundred nodes in this model, and close to as many network connections and needlines. So, we specify an ordered sequence of a few key Over relations for each needline, and use graph analysis algorithms to determine the shortest path that goes over all the specified network connections in the specified order. This makes model maintenance much more practical, and automates an otherwise tedious process.

NASA has three space communications networks, the Deep Space Network (DSN), Near Earth Network (NEN), and Space Network (SN), all of which have been separately designed, developed and operated for decades. However, new space missions, including, coincidentally, ones such as EFT-1, need to use the services of all three networks, but the interfaces and

services are quite different. The Space Communication and Navigation Office (SCaN) of NASA, which manages these networks, chartered a series of system of systems trade studies to evaluate different architectural options for integrating these networks. The method described in this paper was used in the Cycle 5 and 6 trade studies to document all of the system interfaces, their end-to-end configurations, and their interface protocol bindings [8,9].

Table 2. Example Four Layer Structure

Example	Message	Encoding	Signal	Physical
Document Transfer	Document	PDF file	HTTP stack	Ethernet
Automobile	Stop car	Brake pedal pressure	Hydraulic pressure	Brake caliper pressure

Even more sophisticated techniques to accomplish design and performance evaluation are possible by leveraging the framework described in this paper. Primary among these would be checking the types of the PDU flows to know which horizontal connections between systems are allowable, which end-to-end flows are complete, and the SDU flows within components to know which lower layers may carry data for which upper layers.

We believe that this pattern of interface modeling has wider potential application than just protocol stacks and communications systems. One such area would by systems engineering for physical systems. For example, two components might share a thermal interface whereby they

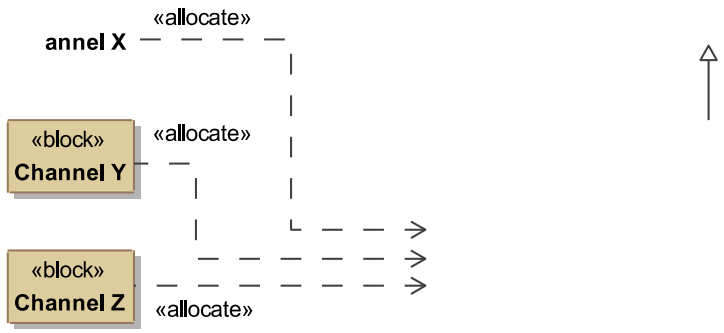

Figure 10. The simplified pattern.

exchange heat. That exchange may be realized by any or all of three different mechanisms: conduction, convection and radiative transfer. This would be modeled as a single connection at the top layer, with three possible layers underneath. Similarly, a copper wire at one layer may be used by several connections at the layer above, such as electrical, mechanical and thermal.

We have done some initial work to show how the layered interface pattern might be applied to more general systems engineering problems. Our pattern, as it exists, does not specify which or how many layers must be defined for a given problem. The OSI model [5] specifies seven layers that are used in computer networking applications. We describe here a simplified,

four-layer model that might apply to systems engineering. The four layers, in order from most abstract to most concrete, are Message, Encoding, Signal and Physical. Table 2.

These examples are somewhat simplified in that they assume a single direct connection at each layer from one end to the other. In a real automobile model, there would be several intermediate steps at lower layers of abstraction. To illustrate this idea see Fig. 6. The top two layers, Command and TC Frame, have no intermediate steps. The ABA Earth-Space Link Terminal and Terrestrial WAN create two and three steps respectively between the two end nodes. The number of layers need not be constant along the entire path. Nonetheless, we feel that these four layers are a good starting point for many system engineering problems.

It is also possible to model how the data representation is transformed between layers. A simple application of this would be to show how telemetry channels are arranged inside a telemetry packet. Such data is commonly put in fixed locations of a fixed-size packet. Fig. 10 shows how channels X, Y and Z are packed in the Message Layer are packed into Packet A of the Encoding Layer. This is a simple description of a data transformation. All packets have sixteen bits. Packet A uses the first four bits for Channel X, the next four for Y and the next for Z. The last four bits of packet A are reserved and unused. The order and position of the data within Packet A is specified by the order and size of its four part properties. The allocate relations here parallel the Over relations to show the correspondence between layers.

Pattern

Ordinary SysML and UML constructs were used as much as possible. The only extensions made were implemented as UML stereotypes to define concepts that are part of the pattern, but not explicitly included in UML or SysML, e.g. «Protocol Entity», «PDU» and «Provided». All of the diagrams in this section are SysML Internal Block Diagrams (IBD). These diagrams show parts connected to each other by connectors. In addition, the ports on the boundaries of the parts are used as the attachment points for the connectors. In this method the ports are further restricted to be SysML proxy ports, which means that they define what data may flow through them, but contain no further structure or behavior. We have defined several stereotypes to help clarify roles and provide a mechanism to support model validation. As specializations of the SysML proxy port, we have defined PDU, Provided and Required ports. The PDU ports may only be connected horizontally. Provided and Required ports may only be

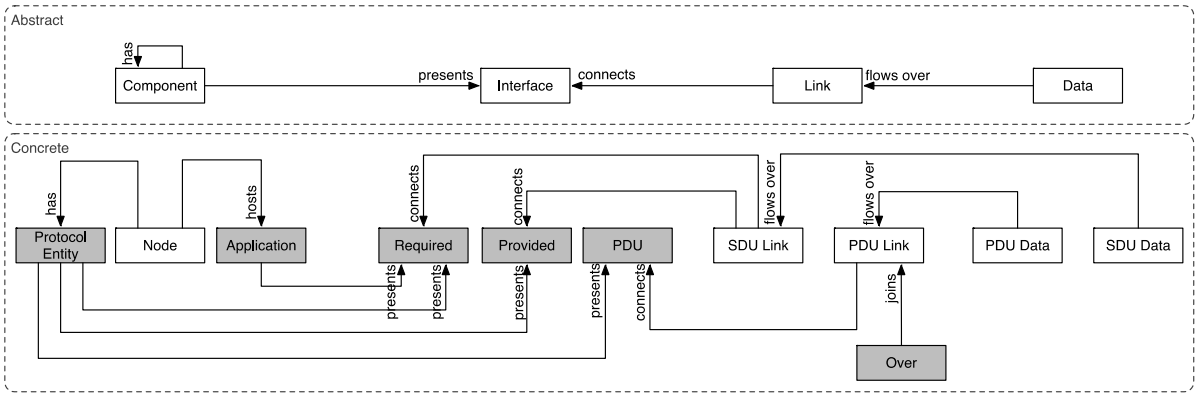

Figure 11. The simplified pattern.

connected vertically, with Provided ports facing up and Required down. The parts inside the systems are tagged with either the Protocol Entity or Application stereotype to further clarify the types of ports they must have. Finally, the Over stereotype used to establish the vertical relationship between horizontal connectors. These relationships must match the vertical connections within the systems.

Fig. 11 shows an intentionally simplified representation of our pattern. This version is not intended to capture all possible configurations of layered interfaces, just a representative set. The pattern is divided into an abstract section and a concrete section. The boxes represent concepts and the lines with dark arrow heads represent relations between concepts. The shaded concepts are those for which we created UML stereotypes in our example. The relationships between the abstract and concrete portions are not shown explicitly, they are indicated by naming and diagrammatic proximity. Neither shown are multiplicities, direction of data flow or other information that would be necessary for a full set of validation and analysis rules.

In the abstract portion, Components present Interfaces, Links connect Interfaces and Data flows over Links. In addition, Components can be made of, i.e. have, other Components. In the concrete portion, there are three kinds of Components. Nodes serve to contain Protocol Entities and Applications. Those two Components are distinguished by the kinds of Interfaces they present. Protocol Entities present all three kinds of Interface, while Applications present only Required Interfaces. SDU Links connect a Required to a Provided interface. PDU Links connect PDU Interfaces. PDU Links may be joined by Over relationships. Finally, PDU Data flows over PDU Links, and SDU Data over SDU Links.

The pattern thus established provides a rich framework on which to hang behaviors, properties, constraints and requirements. The advantage is that these may be applied at the appropriate horizontal place in or between systems, and the appropriate vertical level of abstraction. For example, it would be appropriate to put statements about the raw, low-level bit rate on the RF connector between the ESLT and the space user node. It would not be appropriate to place it on the top-level Command connector. However, given the bit-rate constraint and the relevant protocol layers in the stack (and their PDU specifications), it would be possible to compute the effective bit rate of commands, excluding the overhead that the lower layers add. Alternately, the analysis could be run in reverse, given the required number of commands per second, and the sizes of commands, that could be compared to the bit rate to see if the RF connection can support the desired behavior. Once populated with these specifications and constraints, the framework can be used as the basis for a variety of analyses, including static, dynamic, simulation, validation, etc.

Future Directions

There are several areas in which we would like to see this basic approach extended. The OSI BRM defines seven abstract layers of communication between open systems. It then assigns functionality and requirements to each layer. It says nothing about any specific network protocol but it gives protocol designers and implementers a framework within which to work. We think it would be beneficial for other domain practitioners to devise similar layered descriptions of interfaces in their own domains. This could then help implementers to build systems in those domains, and systems engineers to integrate such systems. Domains such as electrical, thermal, and fluids systems also exhibit similar concerns where separating out external, internal, logical and physical viewpoints might prove to be of use in describing these systems.

The viewpoints, views and concern we have defined up to this point are high level. We would like to encourage the development of other viewpoints. Some areas of particular interest would be views for networking, relaying and tunneling. We have already addressed bridging in a simple case like the ESLT. A more general approach would identify different relaying opportunities in a complex network of interfaces. Tunneling is when the usual vertical order of interfaces is broken and a low-level data flow is transferred over what is normally a high level connector. This has found many applications in computer networking, notably virtual private networks.

The mapping between layers in general is many-to-many. A single upper layer between two systems may have several different transport mechanisms below available to it. The signal to apply the brakes may be encoded as hydraulic pressure or an electrical signal, depending on the design of the car. Or, the same Ethernet cable can transport FTP, HTTP and many other kinds of content. It is also possible for the same upper layer to be routed through different intermediate systems at lower layers. A commercial airline flight from New York to Los Angeles may be routed through either Chicago, Denver or Dallas. The future work in these areas would be to develop analysis and validation algorithms for these applications, and make corresponding extensions to the pattern where needed.

In our early experience, there seems to be a conceptual divide between how different people put emphasis on different parts of the pattern. More work needs to be done to bridge this gap, and extend the pattern to communicate better to all. System engineers typically work with physical rather than information systems. As such, they seem to resonate more with the PDU exchanges between Components. This is naturally what systems engineers are concerned with. An important part of the future work would be to find a metaphor for the transformational SDU exchanges that happen within Nodes in domain of physical systems.

At a pragmatic level, there is much to be gained by integrating UML and SysML based model with external analysis tools and standards. This is a common pattern in Model Based Systems Engineering. It is not generally intended that the system engineering modeling tool perform specialized analysis. Rather, it generally serves as a central repository that federates with more domain-specific and detailed tools. The forms of this analysis could be static, or dynamic, especially tools that perform discrete event simulation. We also foresee the beneficial use of more and different graph analysis algorithms. Our applications to date have used a shortest path analysis, but other algorithms could be used to determine the amount of redundancy in the connections between systems, the points of greatest vulnerability and end-to-end performance.

Summary

We have described a general-purpose pattern for modeling interfaces that has a wide range of applications. It leverages a well-established pattern from the Open Systems Interconnection Basic Reference Model, in conjunction with model-based systems engineering, and the analysis of stakeholders, concerns, views and viewpoints as described in ISO 42010 and RASDS. The primary benefits of this pattern are in the way it lets system engineers separate concerns in their design. By treating the layers of abstraction separately, it allows more specific definition of required interaction between systems. To the extent that stakeholder concerns can be associated with particular levels of abstraction, it enables the production of views that are more narrowly focused on those concerns. Similarly, it enables the explicit specification of the transformation of data or material as it passes vertically between layers. Individual views slice the system in different ways to show overview or details. It enables end-to-end data flow analysis and addresses interactions at each layer or combination of layers. In the current state of practice, we have observed that different layers of abstraction are easily conflated, leading to complexity in presentation of views, and potential gaps and conflicts in the design. We hope to extend and apply the pattern to other domains of engineering and system engineering to address the increasing complexity and number of interfaces that system engineers must manage for today's projects.

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

Acknowledgements

The authors would like to extend special thanks to Sanford Friedenthal for his feedback and guidance during the writing of this paper.

References

1. ISO/IEC 42010.
2. Reference Architecture for Space Data Systems (RASDS), CCSDS 311.0-M-1
3. OMG Unified Model Language™ (OMG UML), Infrastructure & Superstructure, Version 2.4.1.
4. OMG Systems Modeling Language (OMG SysML™), Version 1.3.
5. ISO/IEC 7498 OSI BRM.
6. Space Communication Cross Support Architecture Description Document (SCCS-ADD), CCSDS 901.0-G-1.
7. Fosse, Delp, "SE Interfaces," IEEEAC Paper #2562, 2013.
8. Shames, et al., "NASA Integrated Network Monitor and Control Software Architecture", Space Ops 2012 paper #127528, 2012.
9. Tai, et al., "NASA Integrated Space Communications Network", Space Ops paper #1275818, 2012.
10. Jackson, et al., "Architecting the Human Space Flight Program with Systems Modeling Language (SysML)", Infotech 2012, AIAA 2012-2556.

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). He currently is the System Architect for the SCaN Network Integration Project (SNIP), using MBSE techniques to define how to integrate the three NASA space communication networks (DSN, NEN, SN).

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-four years.