Everything you wanted to know about interfaces, but were afraid to ask

Louis S. Wheatcraft
Compliance Automation, Inc.
1221 South Main Street, Suite 204
Boerne, Texas 78006-2836
Phone: (830) 249-0308
E-mail: louw@complianceautomation.com
Internet: http://www.complianceautomation.com

Abstract. Some of the biggest problems in developing a system are at an interface. Some of the most critical requirements for every system that we build are interface requirements. Interface requirements cannot be written in a vacuum, both sides must participate. Yet how to write interface requirements is barely covered in the literature – and what is in the literature is not consistent. This paper will address some things you can do to get better interface requirements. Topics covered include: Understanding what constitutes an interface, how to identify interfaces, how to define and document interface definitions, what constitutes a good interface requirement, and where and how to document interface requirements.

Why are Interfaces Important?

Systems are part of other systems, are made up of subsystems, and interact with other systems no matter what level your system of interest (SOI) is at. This applies to simple as well as complex systems, hardware systems, software systems, and hybrid hardware/software systems. The interactions between your system and others are called interfaces.

Identifying interfaces helps you to define your system's boundaries. Indentifying interfaces also helps you understand the dependencies your system has with other systems and dependencies other systems have with your system. Failing to identify an interface can have unpleasant repercussions on your project and is a common reason for products that fail to meet stakeholder expectations. Missing or incorrectly defined interfaces are often a major cause of cost overruns and product failures.

Indentifying interfaces helps you ensure compatibility between your system and other systems you need to interact with. Many projects neglect to identify and control their interfaces until testing. The first encounter with the results of this oversight often occurs when people find out that they cannot connect test equipment to their system to perform the tests. Worse yet, think of the problems when you turn your system over to operations and a missing interface is discovered such that your system cannot function or another system depending on your system cannot function. By identifying and managing your SOI's external interfaces early, you are identifying key drivers for your product that must be addressed in your system requirements.

Identifying interfaces also helps to expose potential problem areas and risks to your project. There are often existing systems that your system has to interface with that are established and cannot change – this might be a problem for you – it might not, but you need to know. There may be systems that your system has to interface with that don't currently exist that are being developed concurrently with your system. How can you develop requirements for your system when you don't know what the interfaces are or the characteristics of those interfaces with those other systems? You need to know any issues associated with your interfaces early so that you can insure compatibility with existing systems or work with the other developing system to jointly define the interfaces so you are compatible.

Serious problems can and do arise at interfaces due to the inherent risks associated with a system's interfaces. Because the interfaces represent systems outside your control, your system is vulnerable at your interfaces. If an interface is not well understood, not defined, or is subject to change, your system will be impacted. There is also the threat of someone outside your system impacting your system's performance – either intentionally or unintentionally. There is an old saying "If you want to sabotage someone's system, do it at an interface."

Because of the importance of identifying, defining, developing interface requirements, and managing these activities, interfaces need to be a prime concern of the project System Engineer, lead Software Engineer, Business Analysis, or anyone else involved in developing requirements.

Given the importance of interfaces, you would think that there is a standard process to indentify and define interfaces, to develop interface requirements, and manage these activities. Unfortunately there is not. Given the different cultures within industries and within organizations, each manages these activities differently. This results in a lot of confusion on where to document this information and even what these documents are named.

Regardless of the names we give various documents that contain information concerning interfaces, there are some guiding principles and best practices you can follow. These best practices are based on lessons learned as a result of being exposed to a variety of approaches to managing interface requirements and having seen what approaches work best and what approaches that tend to lead to problems.

What Is An Interface?

"An interface is a boundary where, or across which, two or more parts interact." Another definition is: "An interface is that design feature of a piece of equipment that affects or is affected by a design feature of another system." This interaction is shown in Figure 1.

The key words here are "interact" and "affects or is affected by another system". From a requirements standpoint, any time the wording of a requirement indicates or implies one of these conditions, there is an interface involved. If there is an interface involved, then the requirement dealing with this interface is classified as an interface requirement.

If there is an interface involved, it must be defined and mutually agreed to by the systems on each side of the interface. Your interface requirements must reference this agreed to definition as well as the

interface requirements on the other side of the interface. Where the interface definition is documented and how your interface requirements reference this definition depends on the maturity of the system on the other side of the interface. This will be discussed in more detail later in this paper.

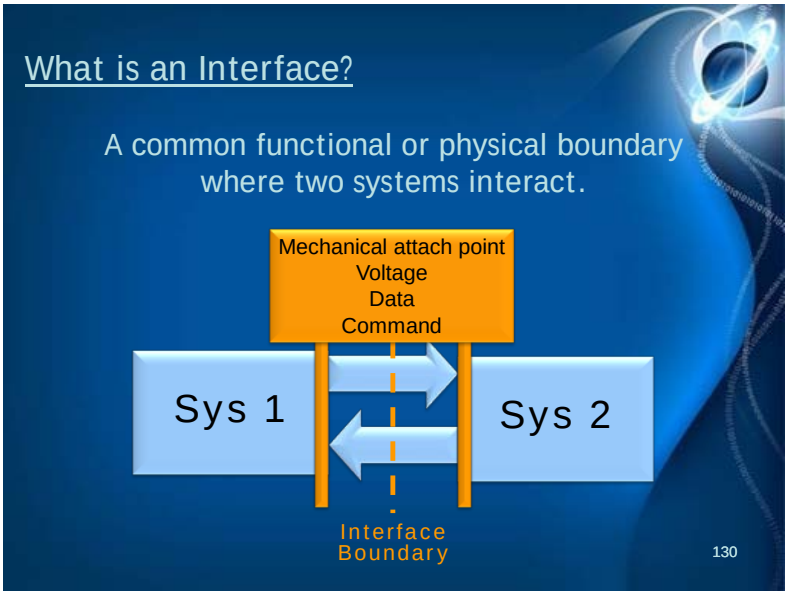

Figure 1: Definition of an Interface

It is also important to understand what an interface is not. An interface is not: a subsystem, a component, or a function. The hardware and software on each side of the interface is responsible for meeting the interface as defined in an interface definition document. They do this by meeting the interface requirements documented in their system's requirement document that include a reference the agreed to common definition. Figure 2 shows examples of what an interface is not.

Figure 2: Examples of what an interface is NOT.

Unfortunately, we frequently see statements like these. The first requirement assumes the interface is a system and has functionality – this is not true. The second is a requirement on the designers and also assumes the interfaces are things. The requirement should be on accessibility of connectors, bolts, etc. The third and fourth again assumes the interface is a thing. These are requirements on each of the systems and apply to any hardware or software of the system involved in interfacing with another system. The bottom line: There should be no requirements that say "The interface shall ….."

Process to Write Interface Requirements

Writing interface requirements is a three-step process:

Step 1: Identify the interfaces

Step 2: Define the interfaces

Step 3: Write the Interface requirements.

It is important to understand that, like Systems Engineering, writing interface requirements is an iterative and recursive process. These three steps are repeated for each level of your system's architecture. There are new interfaces (internal) that need to be identified and defined. Interface requirements must then be written and documented for each SOI that makes up this architecture. Another important point is that your knowledge about the interfaces evolves along with the system design. Your knowledge increases with the design, until actual drawings are available showing connector part numbers, pin assignments, and mechanical interfaces.

Step 1: Identify the Interfaces

This first step involves an analysis of your SOI and the context in which it relates (interacts) with the parent system it is part of (external interfaces) and an analysis of the parts that make up your SOI and how they relate (interact) with each other (internal interfaces.)

The tools that are often used as part of this analysis are Operation Concepts, System Block Diagrams, N-Squared (N2) diagrams, Allocation Analysis, External Interface Block Diagrams (Context Diagrams), and Interface Block Diagrams. The intent is to define your system's interfaces top down. Start with the Parent System Block Diagram, then develop an N2 diagram to refine your knowledge of the interfaces between all elements that make up your parent system including the interfaces of your system. Then using this knowledge along with the knowledge from your system's Operational Concepts, develop an External Block Diagram for your system showing all external interfaces of your SOI with all other systems, for all lifecycles. Then for each system you interface with, you do an individual Interface Block Diagram, showing all the interfaces between your SOI and that system. Once you have defined the architecture for your SOI, this approach is repeated at the next level to address your system's internal interfaces.

Operational Concepts: Your system will have different interfaces at different times in its life cycle. All must be considered to make sure that nothing is overlooked. An effective way to identify interfaces is to develop Operational Concepts from different stakeholder viewpoints addressing each lifecycle stage. The stakeholders associated with each of the systems your System's interfaces with will have unique knowledge you need when identifying and defining interfaces and their associated interface requirements.

System Block Diagrams: System Block Diagrams (SBDs) show all architectural elements of the parent system (of which your system is a part) and their interfaces. SBDs are usually fairly high level (mechanical, power, commands, data, human, etc.) that show both internal and external interfaces of the parent system. The directionality of the interaction between systems can be shown with arrows. The SBDs give a "big picture" view, showing both internal and external interfaces of your parent system and your SOI's place within this architecture.

N2 Diagrams: N2 diagrams (sometimes referred to as I2 diagrams) allow you to systematically compare each architectural element with every other architectural element that makes up your parent system. You can start with an N x N matrix, where N represents the number of subsystems of your parent system, to identify your relationship with the other elements of your parent system's architecture (external interfaces) and then develop an N2 diagram of your SOI to show the internal interfaces between your subsystems or components that are part of your SOI's architecture. The N2 Diagram is used to identify general classes of interfaces (mechanical, power, commands, data, human, etc.) and is very helpful to help make sure all interfaces have been identified and not missed. (An example N2 Diagram can be found in the latest version of NASA's System Engineering Handbook, SP-2007-6105 Rev 1, Figure 4.3-4 on page 54.)

Allocation Analysis: When a requirement at one level is allocated to two or more elements of a system's architecture at the next lower level, there may be an interface between those elements. In order for each element to do its job, it may require an interaction between those elements. If there is an interaction, there is an interface. Allocation analysis is an important tool to use in identifying interfaces.

External Interface Block Diagrams (EIBD)(Context Diagrams): An EIBD shows your System and its external interfaces to other architectural elements at the same level and external interfacing systems for all lifecycle stages of the your SOI. You need to show more than just the interfaces of your system during nominal operations, you must also include interfaces for all of your SOI's lifecyles, including development, testing, verification, transportation, handling, servicing, and maintenance. All of which should have been addressed in your Operational Concept and in your N2 Diagram. Section 3.1 of your System Requirements Document (SRD) should include your system's EIBD so that readers of the requirements understand the boundaries of your system and understand where there are interfaces with other systems. Your SRD will contain individual interface requirements for each of these interfaces.

Interface Block Diagrams: Interface Block Diagrams are developed, one for each of the systems shown in the EIBD. Your SOI is shown on one side of the diagram and the other system on the other side. In between, you show all the interfaces (interactions) between your SOI and the other system. Start with high level information (mechanical, power, commands, data, human, etc.) and then further refine each of these (instead of just power, 28 VDC; instead of just data, include the general types of data. Each of these interfaces will then be "defined" using textual statements, diagrams, tables, drawings, graphs, and figures. These definitions are what is documented in your interface definition documentation. There will be one Interface Block Diagram for each of the systems your SOI interfaces with as shown in the EIBD.

The System and External Interface Block Diagrams should be included in your Operational Concepts Document and SRD to show the reader the "big picture". The N2 Diagrams, External Interface Block Diagrams and Interface Block Diagrams should be included in your interface definition documentation.

Step 2: Define the interfaces

Once an interface has been identified, it needs to be defined. To define an interface you need to define the characteristics of each system at the interface, the media involved in the interaction, and the characteristics of the thing crossing the interface.

The characteristics of the system at the interface could be an electrical, electronic, or fluid/gas connector or a mechanical interface where the two systems are bolted together. What your system looks like at the interface may be documented in a drawing showing the mechanical interface, bolt hole patterns and sizes, thickness of the metal, characteristics of the mating surfaces, etc. If an electrical, data, or command interface, the connectors involved with the connection: type of connector, pin assignments, isolation, grounding, etc. If a fluid or gas connection, the allowable leak rate, if any, needs to be defined both during mating and while mated together. If interfacing with an existing system, this information will be known from the start and documented in some form of interface definition document. If the system you are interfacing with is being developed concurrently with your system, this

type of information may not be known and documented until the design is complete. This is discussed in more detail later.

The media involved could be electrical through a wire, physical contact, fluid or gas flow though plumbing, an RF signal through the air or space, fiber optics, data via a common communication buss or the internet. To define the media, wire sizes, types of wire, pipes, pressure ratings, burst pressure requirements, leak rates, etc. need to be defined and agreed to. In some cases a standard defines the connection, the media, and transport protocol, for example, Ethernet or USB. Again, this information may not be known until the design is complete.

The thing that is involved in the interaction could be electrical, in which case you define the characteristics of the electricity involved: voltages, currents, noise, impedance, ripple, rise and fall times, etc. Included would be any requirements concerning protection from shorts and current spikes. It could be commands, data, gases, hydraulic fluid. In the case of commands and data you would have to define what commands, what data, and their format and identifiers. If gases or fluids, you would identify what quality and quality these must be along with pressure, temperature, and flow rate. Often this is identified as a standard the gases or fluids must meet. This information must be defined and agreed to before design so the designers can design each system to meet this agreed to definition of the interface.

A key point is that all this detailed information must be defined and documented somewhere – no matter what you call the document. For developing systems, not all this information will be known until the design is complete. In the beginning all you may know is what the thing is that is involved in the interaction and its basic characteristics. The details of what your system looks like at the interface and the media involved may not be known until the design is complete. Thus you start with place holders – To Be Determined/To Be Resolved (TBDs/TBRs) - for tables, drawings, graphs, etc. When the information is known, you fill in the TBDs/TBRs. Before the systems can actually be built, there can be no TBDs/TBRs in the definition document, whatever you call it.

The general format of the definition statements is:

- [Thing being defined] is[are] [whatever the definition is]. or
- [Thing being defined] is as shown in [Drawing xxxxxx] or [Figure yyyyyy]. or
- [Thing being defined] has the characteristics shown/defined in [Table or Graph zzzzzzzz.] or
- [Thing being defined] is as defined in [Table or Graph zzzzzzzz].

Notice the terms “shall” and “will” are not used in definition statements because you are just stating an agreed to fact.

Examples include:

- The DC voltage [supplied by System 1] has the characteristics shown in table xyz or figure 123.
- The mechanical attach point [between System 1 and System 2] is shown in drawing xyz..

- The fluid [supplied by System 1] has the characteristics defined in table xyz (pressure, quality, flow rate, temperature, etc.)
- The leak rate at the connection [between System 1 and System 2] is less than xxx.
- The commands [sent by System 1] are defined in table 123.
- The data stream [accepted from System 2] has the characteristics defined in ……..

These are statements of fact that need to be written down and agree to.

Defining Interfaces for Existing Systems

Existing systems will have their interfaces defined in the form of tables, figures, and drawings. Outputs of the design process. In the example above, if System 1 exists, the owner of the system will have documented these interfaces in a configuration controlled document so others, System 2, can be designed to interface with System 1 per that definition. In this case, System 1 does not know of System 2's existence. System 1 has no requirements to interface with System 2 specifically, only requirements to provide for an interface for some undefined system. From System 1's viewpoint, if any other system wants to interface with it, that other system has to do it per System 1's rules [interface definition]. System 2 will write interface requirements in its SRD that point to the System 1 document where the interface is defined. (Details on writing interface requirements are discussed later.)

In my experience, the place System 1's interfaces are defined are frequently documented is in an Interface Control Document (ICD). The ICD is developed, owned, and controlled by System 1, the existing System. I have seen others use an Interface Design Document (IDD) in place of an ICD. Others may include this information in a Technical Data Package (TDP). I have seen software definitions defined in Interface Agreement Documents (IADs), Application Programming Interface (API), or Interface Definition Agreement (IDA). I have found that there is no real standard for how to do this and what name is given to the document that contains the interface definitions. What you call the document where you document the interface definitions doesn't matter. What matters is that the interface definitions are documented.

Whether you call this document an ICD, IDD, TDP, IAD, API, IDA or something else, this document documents the interface definition for the existing system, reflecting its "as built" configuration. This document is owned and controlled by the existing system. This document defines what developing systems must do in order to interface to this existing system. Any developing system that wants to interface with the existing system must comply with the interfaces as defined in the this document. The developing system will include requirements in their SRD that refer to the this document for the interface definitions. This document does not contain "shall" statements because the ICD or similar document is defining the as-built system interfaces – statements of fact.

Defining Interfaces For Two Systems Being Developed Concurrently

For the case where both systems are being developed concurrently (neither system currently exists or both are being upgraded), things get more complicated. The interface definitions between the two

systems will evolve over time as the design matures. In the beginning, you are working in the problem space, and your focus is on "what" not "how". Concerning the interaction between the two systems, what information do you need to define and can define so that the design team can design the interface? What do the designers need to know to do the design? What do you know that you can tell the designers ("what" not "how")? What guidance do you need to communicate to the designers? What are the constraints?

Once the design is complete, then, like the case of an existing system, the interfaces will be defined in figures, tables, and drawings. Predesign, these definitions need to be documented and controlled somewhere. For systems that are being developed concurrently, the organizations responsible for these systems must mutually agree on the definition of the interface and document these definitions. From a control standpoint, the common parent to both systems controls the document. There can be multiple definition documents to address interfaces at various levels of your architecture.

Where Should Interface Definitions Be Documented?

Predesign, the [System 1] to [System 2] interface definitions define the functional definitions for all aspects of the [System 1] to [System 2) interfaces. These details flow from the Interface Block Diagram for System 1 and System 2. This includes definitions related to mechanical attachments, loads and structure; power characteristics, environments; gases and fluids; communications, data and information; guidance and navigation; and human factors necessary to design the systems for successful integrated operations. These definitions include the functional interface definitions and their associated rationale. Like the interfaces definitions for existing systems, the definition document does not contain "shall" statements because it contains only interface definitions – statements of fact – not requirements. The functional and performance related interface requirements in each systems SRD will point to these common, agreed to definitions.

As stated for existing systems, interface definition documents for concurrently developed systems have a variety of names. Some organizations put the predesign definition information in an ICD part 1, and the post design details in ICD part 2. Others put the predesign information in an Interface Requirements Document (IRD), an Interface Definition Document (IDD), an IAD, IDA, or a Data Dictionary. (Using IDD when it has two different names can be confusing. Also, the name IRD is problematic in that it has the word "requirement" in the name so people feel that the document has to have requirements (shall statements). If this happens, then you have to address allocation and verification of these requirements. This can be a problem. See the comments later about where to document interface requirements.) I recommend that the document you use to define interfaces, contain only definitions with no shall statements. The interface requirements that point (or linked) to the definitions should be documented in your SRD along with all the other system requirements.

The knowledge needed to completely define these interfaces will evolve with the system designs of each side of the interface. As you move down the levels of the system architecture (down the "Vee"), you will be able to add more detail to the definition in your interface definition document. Each developing system's SRD requirements dealing with an interface defined in the definition document will refer to the applicable definition in the pre-design interface definition document. Requirements contained in lower

level requirement documents will point to more detailed (specific) definitions that apply to that element. Interfaces must be identified and defined for each level of the architecture.

The pre-design definitions do not define the detailed design implementation of these definitions (e.g. electrical connector part number, location, pin assignments; etc). Like the case of an existing system, post design definitions, these design implementation details will be located in the [System 1 to System 2] post-design ICD, IDD, TDP, IAD, IDA, or other similar type of document. It doesn't matter what you call this document, as long as it exists and contains the definitions.

Again, neither the predesign nor post-design interface definition documents contain "shall" statements because they contain only interface definitions – statements of fact – not requirements.

Post-design interface definition documents establish, define, and control the detail interface design and implementation between two interfacing systems. This document contains the detailed interface design drawings, definitions, characteristics, and constraints of the interfacing items. The End-Item Specifications (EIS) for your system contain post-design requirements that are passed on to manufacturing. The post-design interface definition documents are included in this package, which some organizations refer to as a Technical Data Package (TDP). Some organizations tie all the post design information together in an ICD which is part of the TDP.

Do I always need a separate document for interface definitions?

You don't always need to document interface definitions in a separate document. The defining factor is organizational involvement and control. If each system being developed is managed by different organizations (vendors), developing a separate document for interface definitions is a good idea, as it serves as an agreement (contract) between the two organizations on how the interfaces are defined. In this case, each of the children systems interface requirements will point to the definition document for the interface definition as shown in Figure 3.

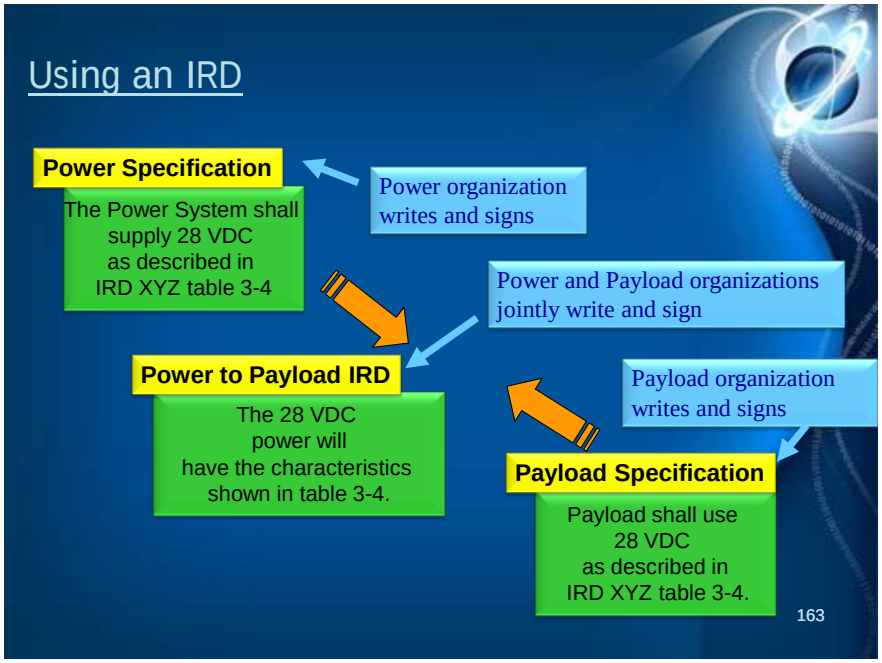

Figure 3: When the interface is defined in an IRD

However, if your SOI is an element of a higher level system, the other system is also an element of the same higher level system, AND the development is being done within the higher level system's organization, the definitions can be contained in the parent system's SRD. In this case, the children systems, interface requirements in their SRDs will point to the parent document for the interface definition as shown in Figures 4.

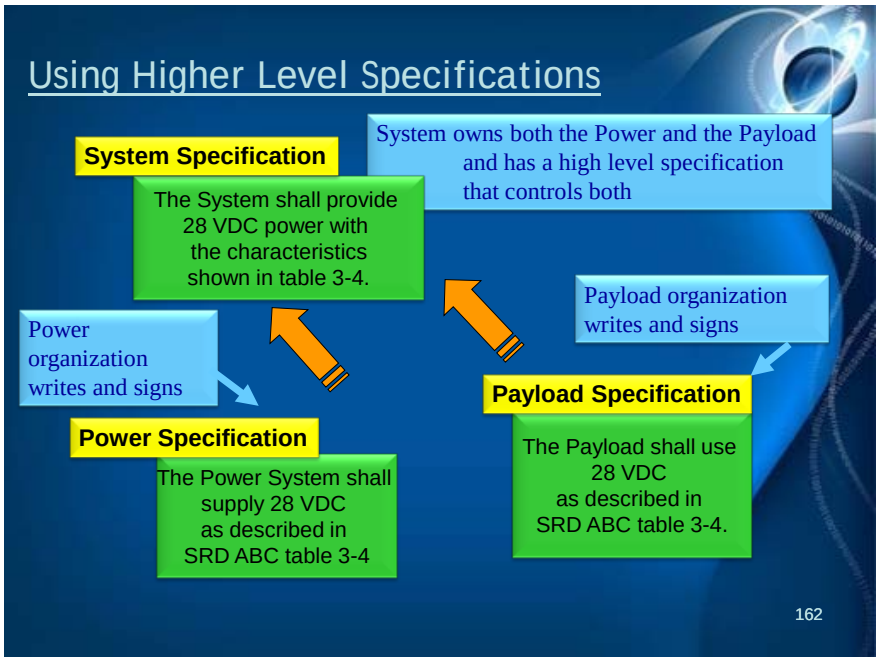

Figure 4: When the interface is defined in the parent system's SRD

Step 3: Write the interface requirements.

When the interfaces have been identified and defined, you are now ready to write interface requirements. (Following this order is an ideal case. I have seen many cases where the interface requirements are written before the interfaces are defined. In this case, the interface requirements point to a TBD document that will contain the interface definition.)

An interface requirement is a system requirement that involves an interaction with another system. The format of the interface requirement is such that it includes a reference (pointer) to the specific location in the definition document that defines the interface. Interface requirements are written in pairs as shown below and in Figure 5 on the next page. All interface requirements have the same general form:

> "[System 1] shall [interact] with [System 2] [as defined in or having the characteristics shown in] [a section of the document that defines the interface]."

> "[System 2] shall [interact] with [System 1] [as defined in or having the characteristics shown in] [a section of the document that defines the interface]."

In this example, we will verify that the System 1 was designed to interact with System 2 per the definition in the document that defines the interface and that System 2 was designed to interact with System 1 per the same, agree to interface definition.

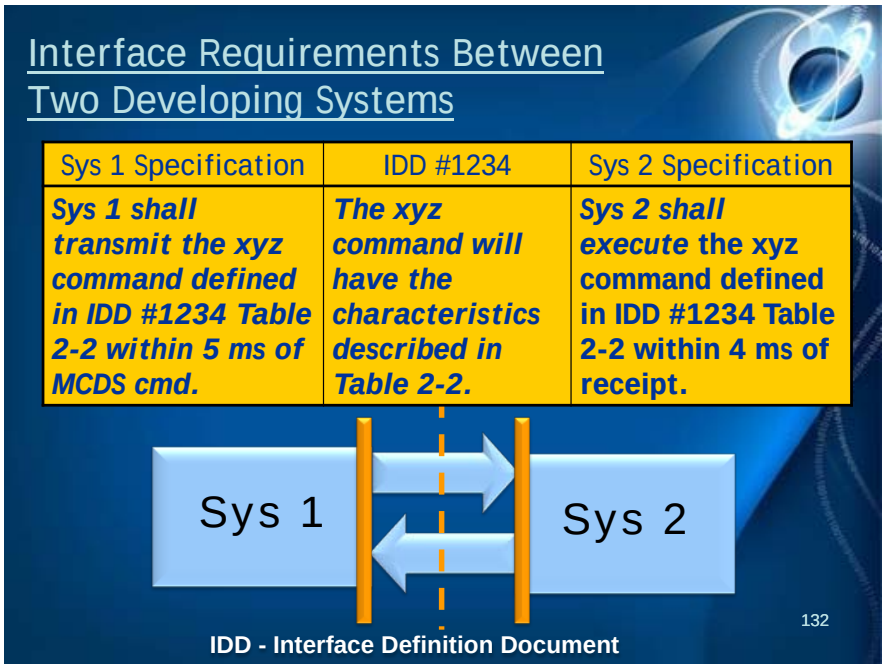

Figure 5: Pairs of Interface requirements pointing to an IRD for a common interface definition.

The word interface is not included in an interface requirement. A requirement that says "[System 1] shall *interface* with [System 2] as defined in [document that defines the interface]" is ambiguous. There could be multiple interactions between System 1 and System 2. There needs to be a separate interface requirement for each of the interactions. Doing this helps when you allocate the interface requirement to the next level of the architecture. It also addresses a basic characteristic of good requirements: keeping the requirement as a single thought. This will help when it comes to verification.

Concerning verification, the interface requirements in your SRD have a "shall" and therefore is what is verified – not the definition statement in the document that contains the definition. What needs to happen is for each side of the interface to verify that its design was per the agreed to definitions in the *common agreed to* definition document and then the parent requirement verification will verify the functionality and performance across the interface. I was told of an example where two mechanical systems that had to be bolted together were being concurrently developed by separate contractors. Rather than having a common interface definition document and drawing each system was built to, each organization had their own, separate drawing. At the parent system Preliminary Design Review (PDR), it was discovered that the number of bolt holes in the two drawings were different!!!

Where Should You Document Interface Requirements?

We advocate that the interface requirements are documented in each System's SRD along with all the other system's requirements. Each of the interfacing systems will develop their half of the interface pair, with each pointing to the common interface definition, wherever the definition is documented. From a Requirement Management Tool standpoint, each of these requirements will have a link to each other showing a dependency as well as each having a link to the common definition. They will also both have a link to their common parent.

Some organizations put the actual interface requirements in the document that contains the interface definitions, usually what they call an IRD, rather than in their SRD with the other system requirements. We do not agree with this approach.

There are many reasons we advocate the interface requirements be included in your system's SRD.

- It is best to keep all the system's requirements in the same document. Some of a system's functions are internal to the system. However, many of a system's functional, performance, and operational requirements involve the interaction of your system with another system. Why document those requirements dealing with interfaces in a separate document from your other requirements?

- If your system interfaces with multiple other systems, using the IRD approach results in your system requirements being spread across your SRD and multiple IRDs. Why not keep all your system's requirements in one document?

- Like all the other system requirements, your interface requirements have to be allocated to the next level of your architecture. Having the system requirements in multiple documents can complicate this.

- If your system is interfacing with an existing system, that existing system's interface definitions are contained in its ICD, IDD, IAD, IDA, or API. In this case it is logical for your interface requirements to be documented in your system's SRD. Why treat interfaces with systems being developed concurrently with your system any different? Include those interface requirements in your SRD just like you would do for an existing system interface.

- Your interface requirements will be verified just like all the other system requirements. If your interface requirements are contained in an IRD, then you will also have to have the associated verification requirements in that IRD. If there are multiple IRDs, your verification requirements will be spread out across multiple documents. It is better to keep all your verification requirements in one document, your SRD, and have a single Requirements Verification Matrix.

- As stated earlier, when defining interfaces, your knowledge grows as the design matures and lower levels of your architecture are defined. Because of this, you start with a lot of TBDs and TBRs and then fill these in. The maturity of the interfaces matures over time requiring frequent updates to the definitions as your knowledge grows. Having your interface definitions in a separate document with your SRD interface requirements pointing to that document, means that the only document that has to be changed when an interface definition matures or changes is the definition document and not your SRD.

One problem I have seen with IRDs that contain requirements is that the pair is written, but the interface is never defined. System 1 shall supply "W" to System 2. System 2 shall receive "W" from System 1. "W" is never defined!! I have also seen cases where only one half of the interface requirement pair is in the IRD, with the assumption that the one system will coordinate with the other system to make sure the other system "supports" the interaction. These approaches are clearly problematic and can be avoided following the approach outlined above.

Another issue I have come across concerning the use of IRDs to include interface requirements. I have had clients who made the statement "My SRD contains no interface requirements, we put all our interface requirements in the IRD." But when I reviewed their SRD, I found many requirements that talked about an interaction with another system – interface requirements!! I call requirements that involve an interaction with another system, but do not point to where the interface is defined "phantom" interface requirements."

Basics to Defining Interfaces and Documenting Interface Requirements

The basics to defining interfaces and documenting interface requirements are as follows:

1. Define scope and requirements for your SOI. As part of this effort, you will be identifying all your interfaces.

2. When an interface has been identified, you then need to analyze the maturity of the interface and where the interface is defined. If all the systems are in development, document the interface definitions in an ICD, IRD, TDP, IDD, IAD, IDA, API, or similar type document. What you call the document that defines the interfaces doesn't matter. Different product domains, different customers, and different programs use the names differently. Agree on a name at the outset on your project, and be consistent in that usage for the life of the project.

3. Write the requirements dealing with the interface in your SRD referring to the document and section within the document where the interface is defined. You allocate and verify your SRD interface requirement, just like you allocate and verify any other SRD requirement. The requirements for your system (including interface requirements) are documented in your SRD.

    a. Some of the requirements for your system involve characteristics that are internal to your system and some may involve characteristics of your system interacting with another system external to yours. When an external system is involved, you have an interface. When there is an interface involved, the requirement is an interface requirement.

    b. Some of the parent system requirements may have been allocated to both your system and another system. When this is the case, you may have an interface with that other system.

Interface Design Guidelines

Because of the importance of interfaces to your product, there are some basic design guidelines you should follow. These guidelines represent best practices when it comes to designing interfaces. These guidelines need to be considered when defining interfaces, writing interface requirements, and designing your system.

- Minimize the number of interfaces: The fewer the number of interfaces, the less risk.

- Simplify all interfaces: use the KISS principle. The more complex the interface, the more that can go wrong.

- Consider commonality, compatibility, and interchangeability requirements. Does your system have to interface with a lot of other systems either under development or existing? Then you need to use standards rather than design a unique interface.

- Assess interface risks. Assess how volatile the interface is: Design to address change Think about safety: Design to protect your system against any thing "bad" that could cross the interface and damage your product. Consider security, do you need to protect an outside action from an adverse effect on your system? Another consideration is the number of different organizations involved in your interfaces. The greater the number of organizations, the higher the risk.

- Do you need to protect your data? Do you need to control access to your data? Be sure to ask the question, "What is the worst thing that other elements could do to you across the interface?" DEFEND AGAINST IT

Closing Thoughts

- Identify all interfaces – early! Because of their importance, you need to identify all interfaces early in your project to insure compatibility with other systems, define your system's boundaries, and to manage project risk associated with interfaces

- Assign responsibility. Again, because of their importance to your project, you need to assign personal responsibility for each interface.

- Obtain copies of all interface documentation. Understand the differences between interfaces with existing systems and with systems being developed concurrently with your system. All of these interfaces need to be defined and you need to include interface requirements for each interaction between your system and the other systems.

- If the interface definitions are not documented, document them!

- If a new interface, negotiate interface definition, and document those definitions in a common, configuration controlled document.

- Make all interface documentation available to everyone in your organization as well as to the other systems you are interfacing with.

- Involve interface stakeholders in your project. They are the ones with the knowledge and they are the ones you need to work with to make sure the interface is defined.

- Include interface requirements in your system's requirement document.

- Provide traceability between pairs, to parents, and to the common definitions.

- Track/monitor interface change. A change to an interface definition can impact both sides of the interface as well as the parent requirement.

- Follow the interface design guidelines in the previous section.

- Plan ahead for interface verification and compliance testing. Verification can be very expensive. In order to do verification for an interface, you will often need special equipment to measure the things crossing the interface. Also, the system at the other side of the interface may not be available for your system's verification activities or you may not want to connect your system to the other system until both sides of the interface have been tested. To do this you may need simulators or emulators (software and/or hardware). This special test equipment and simulators have to be budgeted for, developed, and verified to a schedule that supports your SOIs verification schedule.

References

CAI 2009, *Requirements Development and Management*, Seminar Workbook, October 2009, Compliance Automation, Inc.

Grady, J. O., *System Requirements Analysis*, Academic Press, 2006

Hooks, I. F. & Farry, K. A. 2001, *Customer-Centered Products: creating successful products through smart requirements management*; AMACOM Books, NY, NY, 2001.

INCOSE 2007, *Systems Engineering Handbook - a guide for system life cycle processes and activities*, Version 3.1, INCOSE-TP-2003-002-03.1 , August 2007, ed, Cecilia Haskins.

NASA, Training Manual for Elements of Interface Definition and Control, NASA Reference Publication 1370, January 1997, ed, Vincent R. Lalli, Robert E. Kastner, and Henry N. Hartt

NASA, *System Engineering Handbook*, SP-2007-6105 Revision 1, December 2007.

BIOGRAPHY

Lou Wheatcraft has over 40 years experience in the aerospace industry, including 22 years in the United States Air Force. Over the last ten years, Lou has worked for Compliance Automation, where he conducts seminars on defining scope, writing good requirements, and managing requirements for NASA, DoD, and industry. Lou has a BS degree in Electrical Engineering, an MA degree in Computer Information Systems, an MS degree in Environmental Management, and has completed the course work for an MS degree in Studies of the Future, Lou is a member of INCOSE, co-chair of the INCOSE Requirements Working Group, a member of PMI, the Software Engineering Institute, the World Futures Society, and the National Honor Society of Pi Alpha Alpha.