Why Should I Care?

It's All About Me

NAVAIR FACE™ TIM Paper by:

Chris Allport (chris@skayl.com)
Sonya Hand (sonya@skayl.com)
Gordon Hunt (gordon@skayl.com)

September, 2017

Table of Contents

Executive Summary

The Open Group's Future Airborne Capability Environment (FACE™) Technical Standard is lauded as a great stride forward in Open Architecture and an enabler of the government's Better Buying Power initiatives. As the FACE Technical Standard continues to proliferate, the distance between the decision to use the standard and the developers who build the systems will increase. There is a tremendous amount of excitement and passion for the standard as evidenced by the active and passionate participation of so many individuals representing so many companies at the FACE Consortium meetings. To maintain this level of enthusiasm, it is helpful for all stakeholders to understand how the standard applies to them in their roles as well as how and why the standard applies to others in the cross-functional team.

In addition to educating, this paper aims to push out the envelope with regard to the way we tend to think of the data model-language aspects FACE Technical Standard. Each section of this paper is intended to capture a generalized view of each represented stakeholder role in order to best convey the spirit of the standard and how it applies to one in that role without delving into the minutiae. As with previous papers, we offer this content as the starting point for conversation.

Why Should I Care?

Introduction

The FACE Technical Standard is very comprehensive. In additional to defining a systems framework and set of business practices, the FACE Consortium is currently in the process of spinning off the Data Architecture Working Group (DAWG) into its own standard. This is a sizeable group with a lot of volunteers working to build a standard that addresses an entire framework for system-of-systems architecture: this short paper will not be so broad. It will focus on the various benefits that can be realized through the use of effective data architecture and data modeling.

Reader's Guide

A whitepaper with a reading guide? This paper is meant to meet you where you are in your current role. As young engineers, we would not have likely cared what an acquisition professional would think about the standard. Our job was to write code – how does that help me? On the other side, as contract administrators, we might not care how a developer is going to implement this thing, but we do need to know that what they implement can be traced to requirements.

That said, everyone's job has a LITTLE_BIT (that's an enumeration measurement with a conceptual type of Amount[1]) of impact and overlap on everyone else's. As a result, we recommend the following:

Start by reading the section of the paper most-appropriate for your perspective. Follow that by reading the other sections to gain an understanding and appreciation for why the standard is important from their point of view.

And without further ado…

[1] That's a data model joke. If you didn't get it, it was intended for someone else.

Business

For someone approaching the "Why should I care?" question from a business perspective, there are myriad considerations and objectives unique to this outlook. Maybe you have to figure out how to keep this massively complex system glued together in a fraction of the budget you used to have. Plus, you keep hearing about how tough the competition is getting, so you want to find that competitive edge, maybe by adding additional capabilities. But of course, when you add capabilities, that's even more work that you have to add to the budget. So, let's get straight to the point and address some of the major business concerns, related questions and explore the benefits of the FACE standard from the business angle.

How can we add capabilities to get ahead in this ever more competitive market?

Let's face it, adding capabilities costs more money. Our proposal estimating systems account for this. We have built systems upon systems for estimating costs so we can minimize risk and maximize profit. A data architecture is not going to change that.

However, it does allow us to make significant gains in other aspects of our work.

It is well documented that the cost of software changes increases exponentially as the product moves through the various stages of development. In short, software is easiest to change during the requirements development phase before any design has been performed or any code has been written. Contrast that to the cost of changing the software after certification, whereby any change in the software drives the need to repeat the test and certification process.

In an appropriate data architecture, the data model can be used to calculate the integration alignment with other similarly documented systems before design begins. This process highlights where the systems are already aligned and where additional effort may need to be focused to achieve required integration. And this is all done before any code is written.

Why Should I Care?

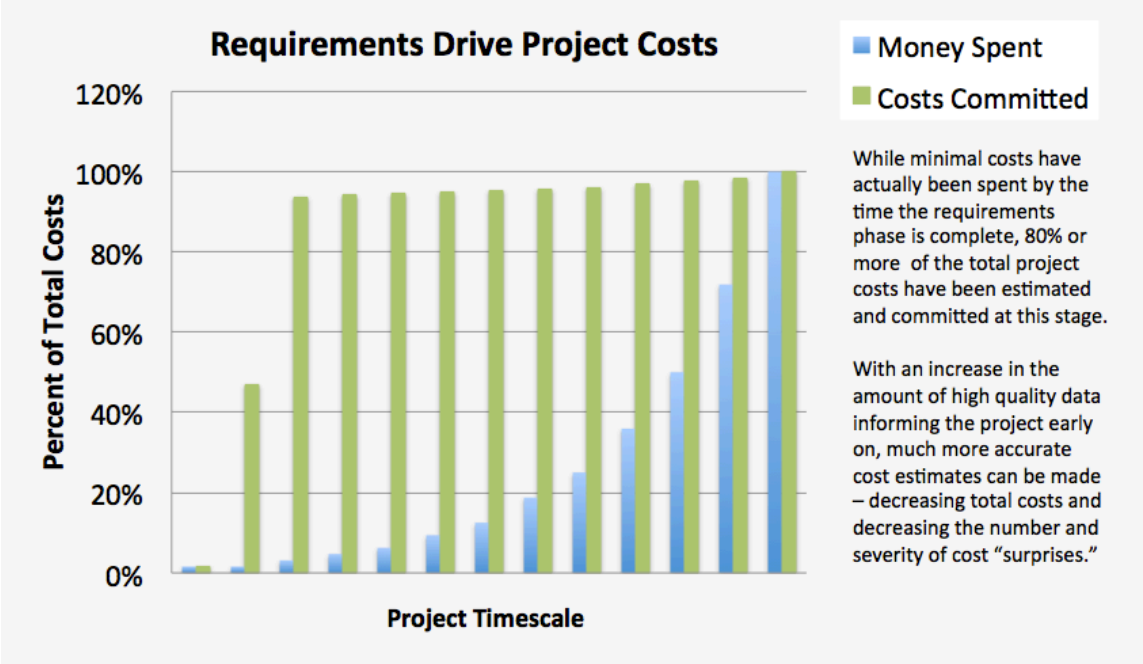

Figure 1 – Money Spent and Costs Committed as a Percent of Total Costs over Project Timescale

It is also possible to leverage the data model to generate data mediation code. Once the process has been validated (i.e. shown to reliably create the expected code), this eliminates the need to manually develop code to mediate between different types of data ever again. This means that portions of code can be automatically generated, eliminating the need for a developer to perform this redundant task thereby reducing cost

Furthermore, Skayl is actively developing a solution that allows for a similar extension to automatically bridge different protocols. With this technology, it is possible to gain an even more significant increase in the capability of automatically generated code. Simply document the protocol and behavior in your data model, and the entire data integration between systems can completely and automatically generated.

By increasing the amount of automation, you are able to eliminate the typical barriers of non-integrated systems and focus the resources you would have previously spent connecting systems on innovation instead.

How do we protect our IP using a data model with open architecture?

Although your data model must document your system interfaces for conformance, you are not required to disclose this documentation. While this may be required by a particular contract, access to the data model can always be managed.

This means that you can freely document your systems and guarantee your customer that the system interface meets the requirements without having to expose all of your intellectual property. Now, you may wish to

share your data model for the sake of sharing the data you collect (or use), but access and use of your data model are always subject to negotiated terms.

Building Data Models is expensive and time consuming. How do we manage?

Designing a good data model that can be leveraged for large-scale integration is difficult. If you have an experienced team specifically familiar with this style of data model, you might be able to budget about 30 minutes to model each attribute, but a safe bet for a skilled data modeling team (though maybe one not terribly experienced with data models organized as specified in the FACE Technical Standard) would be closer to 60 minutes. You can cut this back significantly with appropriate tooling.

Additionally, you can license existing domain data models in specific domains. This alternative provides you with a data model that is supported much like a software development library and has the added benefit of providing an automatic integration capability with others who have licensed the same technology. Furthermore, it opens future opportunities as more domains are added to the data model allowing for the documentation of more and more types of systems. The path to adoption and implementation of a data architecture is not to make it easier to build multiple US models from scratch. How many times do we need to build and document a particular model entity? Each time we build an avionics related component? No, we need only one. The only real path to easing the burden on model providers is to get to the point where they can begin with the 90% solution. And the rest can be integrated into the model with the smallest increment of effort.

Why Should I Care?

Government

Government players have a unique set of goals regarding contracting, costs and concerns around obsolescence. With new technological advancements being made almost daily, there's an increasing need for flexibility, interoperability and automation.

How do we predict and manage costs and changes over time?

Tell me if you've heard this one before: The government issues a contract to a prime. Months (or years) into the project, the government wants to make a change and the contractor tells them it will costs tens (or hundreds) of millions of dollars and will delay the schedule by five years. Sound familiar?

What if, instead of requiring the contractor to stop and perform an impact analysis, they could simply document the proposed change, run their data model integration, and deterministically calculate the overall system level impact? The ultimate goal is to limit the impact of a change such that it is relative to the size of that change rather than to the size of the system.

Instead of a single change rippling out and causing a cascade of changes for which countless additional hours must be billed, imagine that the work could be automatically processed. While this is not yet possible for the physical aspects of a system, it is possible with regard to a system's software interfaces and integration with other systems.

As seen in Figure 2, the cost of a single change increases exponentially with the number of interfaces. However, with appropriate tooling in place, you can see that it is possible to make the cost of a change constant relative to the size of that change regardless of the number of interfaces.

Why Should I Care?

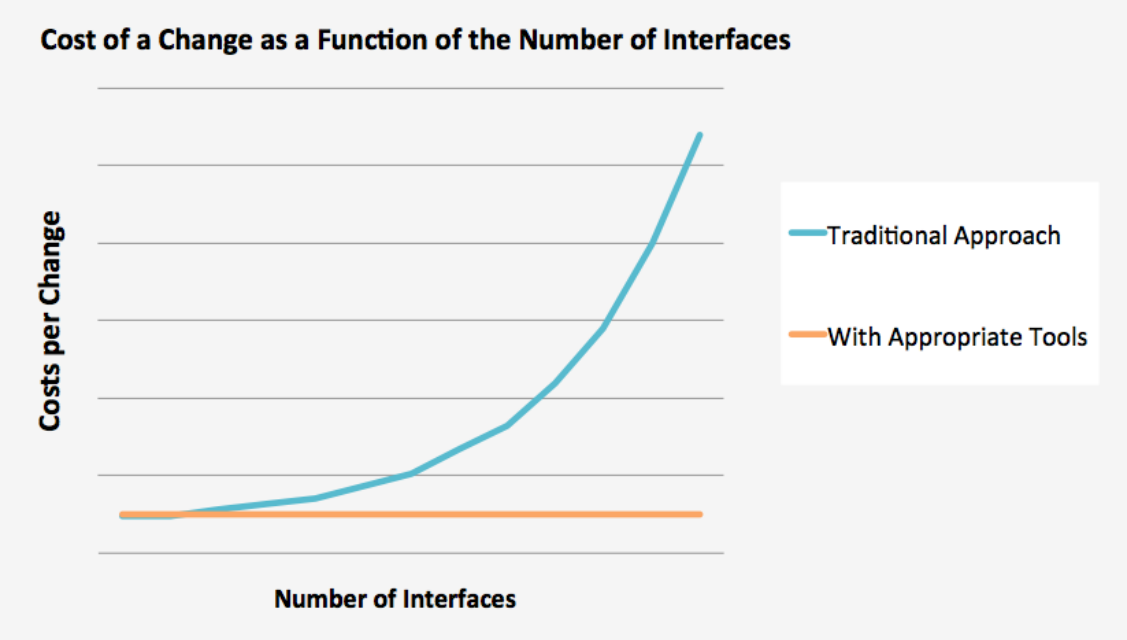

Figure 2 - Cost of a Change as a function of the Number of Interfaces

We typically experience cost spikes at integration time, how can we avoid this?

Avoid surprises!

Last year, we were teaching a class to a group with varying degrees of data architecture experience. While some of these folks were software engineers, many were program managers and other high-level professionals for whom a data model was pure esoterica.

With this audience in mind, we were very careful to craft our question to lead to a single, clear answer. On one particular exercise, all but one person in the class answered the question correctly. This individual was devastated because they thought they were finally starting to understand only to have the rug pulled back out from under them. So, we had them explain their answer.

And you know what? They were correct. That didn't make the rest of us wrong, but they read the sentence with emphasis on a word that, while maybe unconventional, was not overtly wrong. They had used a different set of assumptions about the meaning of the question than their classmates.

Most of our documentation is similarly written.

If we could all interpret all the documentation with the same set of assumptions, integration would be easy. People would understand what we meant to write instead of what we actually wrote. Unfortunately, that is not the case. If, on the other hand, we use the unambiguous nature of data models to capture the semantics of our data, then we effectively reduce the degrees of freedom of our language.

Why Should I Care?

When our interfaces are documented rigorously and unambiguously, we can automatically assess their ability to connect to other systems. When a change (in either system) occurs, it is possible to quickly recalculate the delta. And, in this case, recalculation is truly an automated process. Furthermore, with such rigorously documented interfaces, we are able to analyze their interconnectivity with other systems before design and development begins. This allows us to see where our systems do not align (especially where we expected them to) and allows us to adjust our understanding earlier in the process.

We need to maintain flexibility in contracting. Doesn't the use of a standard restrict my flexibility?

Trick question: what happens when two standards are integrated together and one of the standards changes?

Absolutely nothing. Why? Because the contract was written against a single version of each standard. If the standard changes, the government must negotiate a change order for the new version or else the change is simply ignored.

But what about getting the latest and greatest technology?

We need to rethink how we contract our standards. Ideally, we want a clean way to put all changes to all standards on contract and allow for a very high degree of interoperability across these myriad standards.

It is possible to track changes to a data model and its interfaces over time. It is even possible to leverage this knowledge to connect legacy systems to new procurements. Furthermore, it is even possible to maintain existing software interfaces and continue to reuse them even if the messages change.

The seasoned program manager is jumping up and down in his seat right now, waving his arm back and forth in the air, waiting to be called on by the teacher for having the right answer. "But you can't do that! Not only can you not validate your software, but if the interfaces change, you have no source of data!" And this experienced professional would be mostly wrong. You can do that.

As previously discussed, it is entirely possible to generate valid code. If the integration can be generated and all of the data is present across the entire enterprise, then it doesn't matter if interfaces or messages change – we simply reassemble them as needed with our automated protocol bridging software.

How can we meet the increasing need for interoperability among government players when we're all using different standards?

If Standard A and Standard B are both documented against a data model, then the elements that are common between A and B can automatically be determined.

Do they use data in different formats? No problem. The mediation can be automatically generated.

With a well-developed data architecture, we no longer need to rely on syntax commonality to access each other's software.

Why Should I Care?

How can we increase the reusability of what we're building and manage obsolescence?

Data models are a software product. As with software, data models are not delivered free of defects. Even if the model is correct, it is possible that the documentation used to capture an interface was not. Regardless of the source of the error, the model will need to change in order to address the error.

This presents a problem. How we have built deployed software systems against this data model and now the data model needs to change, doesn't that break existing implementations?

The answer is, quite simply, no. Updating the data model is tantamount to changing an interface control document (ICD). It might change any resulting mediation code that is generated, but isn't that a good thing? After all, the previously generated code was not completely correct.

By maintaining traceability across the versions of our data models, we are able to automatically generate upgrades to interfaces and automatically determine the amount of overlap that exists between standards. Since we can calculate what actually has to be changed, there is no need to change software that doesn't depend on the update.

By clearly and unambiguously documenting our interfaces, we can specify the precise data that our systems need to continue to operate in the future. This innovation changes how we look at obsolescence. We no longer need to update software across the fleet to take some advantage of a software upgrade. We no longer need to refactor software when our underlying communication mechanisms change.

A data model should be managed as another project asset. It is "living" documentation of your system's interfaces. As it matures, it will capture the semantics of your system with increasing clarity, allowing even greater interoperability with other systems. To state clearly, the FACE data architecture results in a model that is not only conformant but also 'usable' and reusable.

Isn't the Data Model just useful for weird and esoteric control systems?

Take, for example, a system based on Smart Health Home technology with many heterogeneous devices and systems. These various devices and systems communicate with one another on a contextual basis regarding the status of the home as well as the status of the home's resident patients. Again, context becomes critically important. The thermostat measures the temperature of the rooms, but the vitals monitor senses the temperature of both Jane and George. With context baked into the data model, the system will provide reporting that is more meaningful to the caretakers, hospital, patient, medical specialists, etc..

Once the data model is built, expansion of the model is accomplished with relative ease. Each new device to be added requires a single unit of effort (see Figure 2) rather than an exponential level of effort based on the number of existing interfaces. In addition, as the model grows, the system gathers new data at an increasing rate, and as such, new patterns may be revealed that may have never before been considered. The data model may even take on a purpose that was not previously envisioned, thus further increasing its usability and lengthening the time to obsolescence.

In the example of the Smart Health Home, patterns in data may result in recommendations (in this case "insights") and the data model may then be used to pass information to direct "behaviors" based on these patterns. For instance, it may be noted that an Alzheimer's patient's high heart rate decreases when lights in the patient's room are dimmed. The data model that contributed to the collection of that data may now be

harnessed to pass information from the heart rate monitor, in combination with the personal location sensor to the occupied room's light dimmers, thus dimming the bulbs in that room to a specified level.

Increases in the usability of the data model are exponentially increased by the added potential future value of the model, essentially changing the game when it comes to obsolescence.

Figure 3 – Concept Interface of Smart Health Home Network View

Why Should I Care?

Figure 4 – Concept Interface of Smart Health Manage Behaviors View

Development

Although the data model utility aids the development team, it really isn't intended for the development team. The development team has some level of interaction with the data model, but this is not simply a UML model from which all of their source code may be generated. That's not to say that this data model cannot be used to generate code, but the scope is very different. This data model is intended to facilitate integration and allows for the generation of integration related artifacts.

Why am I being saddled with a data model? What's the point?

The data model is the documentation of the data in your system or in the system to which you are connecting. Consider it the best documentation of that information you could possibly hope for. Why? Glad you asked.

The data model unambiguously documents the data in the interface. This means there is no need to interpret what the interfaces are supposed to talk about – the data model captures that information clearly. In fact, the documentation is so well-defined, a computer can operate on the data and be used to generate the interface, message manipulation, and mediation (between both data and protocols) software for you.

If you are developing interfaces, the data model needn't slow you down. It is possible for you to build any arbitrary interface and simply document it against the data model. Which comes first? The data model or the interface? It doesn't matter.

And don't worry about your job. There's plenty of work for good developers and even more compelling and interesting work once the mundane is automated!!

How can we ease the pain of adding new systems and making updates to existing systems?

Imagine a world with integration flexibility where there is no longer a point-to-point integration that must be planned and managed. There is no longer a single interface specification to which incoming (or outgoing) data must conform.

Imagine if you could simply rely on getting the data you want and sending the data you have. This is true decoupling of your data interface from your transport interface.

If the format of the messages change, there is no impact on the software. This idea might be easy to accept because we often think about messages getting large with added content, but what happens when messages get smaller and information needed by the interface is moved elsewhere? Once again, since the data interfaces have been completely abstracted, it does not matter how the information arrives. It may matter when the information arrives to allow for coherency of data, but this gets back to the behavioral aspects that Skayl is actively working to address.

But can't I write faster and more efficient software myself?

Perhaps. Small, static distributed applications can always be optimized given enough time and resources. However, today's systems are at such a scale and vary continuously, it would be like trying to manually compile and generate the machine code for a million-line C++ program. One wouldn't consider not using a

compiler. A fully testable data architecture supports the generation of documentation of interfaces that is compilable, simple, and works to any scale.

How hard is it to update my system when messages or interfaces change?

Click a button, regenerate the mediation code, compile, and call it done.

Okay, it may not be that simple, but it is pretty close. When messages or interfaces change, it is necessary to update the documentation (i.e. data model) to reflect those changes. Changing and refactoring a data model can be extremely time consuming. As explained in our previous paper[2], one change at the conceptual level can cause an explosion of changes throughout the rest of the model, particularly as the number of documented interfaces increases. Of course, this impact can be eliminated with appropriate tooling.

I don't understand how to build a Data Model. Where do I begin?

Start with the FACE Shared Data Model (SDM). The SDM is just a collection of common building blocks. These are the things everybody uses. This is the common stuff. This is like going to the toy store and buying a box of Legos™. Just because there is a picture of a car on the front of the box, you can still use the blocks to build a house or a spaceship or a horse[3]. You use the same fundamental blocks to build these different things. If you decide to build with Legos, you don't have to start by building the blocks themselves. You start with an assortment of the different types of blocks. All Lego creations use the same core set of building blocks and, while there are infinite possibilities for how to combine them, any single creation still has a brick-level compatibility with another.

To further extend the analogy, Lego blocks come with a set of rules for how different pieces fit together. These rules are not formalized in documentation but are implicit by the fact that the blocks only fit together in specific orientations.

This is an apt description of the SDM. It is a set of building blocks for constructing all conceptual entities. Just like the Legos, there are rules for how to put things together. As with Legos, one cannot go create new parts that don't fit the previous and follow the rules.

In additional to the fundamental building blocks, the Shared Data Model also specifies the set of allowed systems for measuring each of the building blocks. It is quite a stretch to make the analogy work here, so let's just consider this as a different collection of building blocks that are only used for choosing how we measure (e.g. units and reference frames) the things in our conceptual model.

[2] Data Modeling is Hard vs. Data Modeling is Hard, FACE Technical Interchange Meeting, March 2017, Chris Allport and Gordon Hunt

[3] If you ever successfully built a horse with Legos, please send a picture.

Why Should I Care?

However, just as with our analogy, since our data models (and thus, our systems) are built with the same materials, they have brick-level compatibility.

The SDM only needs to be built once…and you don't have to do it. The SDM is freely available from The Open Group's FACE Standard website.

Now what!?

After getting the Shared Data Model, you have what you need to start building your Domain Specific Data Model (DSDM). It is beyond the scope of this whitepaper to explain how to do modeling (again, refer our previous paper), but it is worth discussing the DSDM a little since it intimately relates to the documentation of your interfaces.

The DSDM captures all the things your system talks about as well as the relationships between these things. It should not look like your class or object model nor should it look like your messages (i.e. message model). It should reflect how things are actually related in the real world and not how we talk about those things.

Data models are also software products. They are not created perfectly the first time but need maintenance and development over time. Data model construction can be a very expensive endeavor. For example, OSD invested nearly $20M in the construction of the UAS Control Segment (UCS) data model.

There are also commercially available DSDMs that you can use as a starting point (or even as the basis for your entire data architecture).

Why do I care about the Semantics of my data?

We've gotten incredibly good at managing the syntax of data. The first real computer term I learned was "syntax error." This is where we tend to start when we develop new standards: we specify a common syntax for everyone to use. Were this a wholly effective mechanism, interoperability would have been long solved by now.

Syntax only gets us a part of the way to interoperability. Just because I can read the data from your interface, how do I know if it means the same thing? We are right back to relying on our ambiguous definitions.

Semantics, on the other hand, is where we capture the meaning of data. And, when we do this with a data model, we are able to capture the meaning in an unambiguous manner because we limit the degrees of freedom. By that, I mean that the semantic is not formed by a sentence or by a disconnected group of tags or words. Instead, the semantic is formed by tracing the relationships of the data as represented in the data model.

I'm not an English major, why do I need to care about Semantics?

Capturing semantics is critical to building a long-lived, truly interoperable data model. For a more in-depth explanation, read the following:

Why Should I Care?

An Audience with the Data Modeling Guru[4]

"Ah, you must have realized that you are having a problem with w-i-n-d." he said.

This was really strange. You know how to spell "wind." Why is this weird old dude spelling simple words to you?

"Don't you mean, wind? As in, here comes the cold front? The blowing of autumn leaves, etcetera?"

"I suppose it could be that, but it rather depends upon your context, don't you think?... What do you suppose your response would have been had you been a watchmaker?"

What? First this guy talks about wind and then he asks me about watches? "What are you talking about?"

"Well, suppose you were a watchmaker and I asked you about w-i-n-d. Do you think you would have just as quickly thought about the weather?" And then, he politely excuses himself and instantaneous disappears.

As a watchmaker, you are concerned with springs and clocks and…ah…winding your watch. He was right, it really is about context.

[4] An excerpt from the soon-to-be-published *Choose Your Own Data Modeling Adventure* by Chris Allport

Integration

A data model can be more than simply a conformance artifact. When applied as part of a well-designed data architecture, a data model can facilitate integration across the enterprise. This is not necessarily a widely held belief – and for good reason. Until now, managing a data model (particularly as it changes over time) has been incredibly difficult and exceedingly expensive. As new tools are coming online to simplify and automate the process, large-scale adoption of data architectures is becoming a reality.

How does a data model enable integration?

Imagine two completely different messaging standards for aircraft. Considering that we have two different standards, they are likely to have slightly different goals or purposes, so we do not expect the two standards to have a 100% overlap but since both messaging standards are focused on aircraft (thus talking about related information in the same domain), we certainly do expect some overlap. Simply, we need only to formally capture the semantics of the messages of both standards and compare them.

Of course, many of us have done this task manually. We have taken two standards and tried to map them to each other. We've done it and it's a painful process (particularly when one or both of the standards is poorly documented). It's time consuming, prone to interpretation error, and invalidated as soon as one of the standards changes. Sure, if you have a subject matter expert in both standards, small numbers of incremental changes can be processes efficiently, but once the number of changes reaches a certain threshold, it is likely to exceed the experts' ability to accurately recall the details of each standard. The main point is that the comparisons have been historically bound by human error.

Once both standards are documented against a data model[5], the comparison are automated. The results are nearly instantaneous (taking minutes instead of weeks or months when processed manually), can be updated as the standards change, and are no longer subject to human error[6].

This same comparison can be used to figure out which messages from one standard match up with messages in the other. And, since the data model captures the logical representation of the data (i.e. units and frames of reference), the conversion from one standard's representation to the other's can be automated (if they, in fact, use different representations).

[5] Since these two notional standards operate against the same domain, they can use the same domain specific data model which significantly eases integration.

[6] Certainly, if the documentation is wrong, the results will be inaccurate – garbage in, garbage out. This capability of dynamic comparison already exists.

Why Should I Care?

How can I streamline the process of adding or extending new capabilities in the future?

Systems change over time. So do standards. Unfortunately, due to a few limitations, deployed systems cannot be updated to the latest versions. Whatever the reason, how do we insulate ourselves from these known limitations?

Design flexibility into the system.

By extending this technology a little further, we can decouple messages from software interfaces. This allows us to truly leverage data from different standards from different transports to work with an interface.

If the messages change, no worries. Run a data model integration utility[7] which automatically finds another source of all the data.

Now imagine using this same principle should you lose communication on your primary channel. With a well-constructed data architecture, you could automatically leverage another source of information.

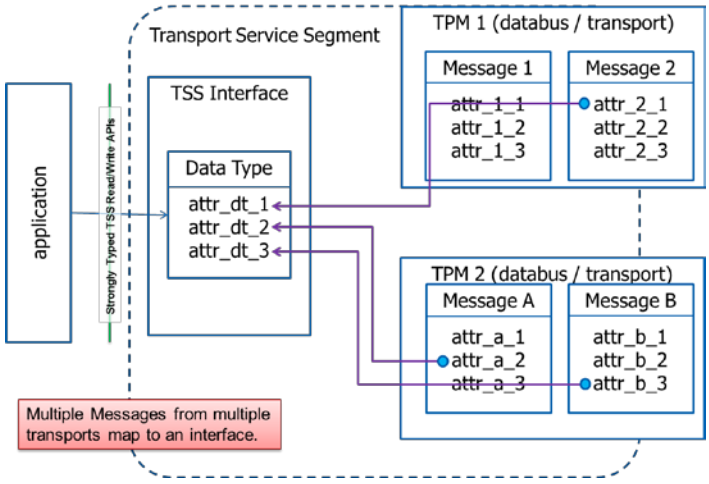

Figure 5 - Interface mapping to multiple transports

How can I improve the process of generating required systems engineering artifacts?

Traditional hand-generated documentation is not automatable, repeatable, testable, scalable. Each update must be made, and then all related and dependent information and documentation must be updated. Also, the set of related information may change based on the updates – essentially the law of unintended consequences.

[7] Yes, this is a real thing.

Why Should I Care?

The challenge of maintaining situationally system awareness of large scale systems in very hard. We specifically design systems in order to minimize the coupling and dependence because we know that chasing down the 'ghost in the machine' during integration isn't fun for anyone.

The data model captures and documents the interfaces and messages with full syntax and semantics. Add the models of interface and communication behaviors and one can calculate where the system data dependencies exist. The model is the documentation, from which system engineering artifacts are automatically generated.

Can I simply promote my messages into Entities to build my Data Model?

Yes, you can. However, you will very quickly find yourself in a world of hurt if you attempt to use your data model for anything other than conformance. If you have a data model refactoring tool, you might find this to be a reasonable start – especially if it is relatively painless to reshape the model into a well-structured entity model.

Consider the following:

Grocery Stores are not Organized by Recipes[8]

"You don't organize your grocery stores by recipes[9]," said the Data Model guru. Then he blinked, and just looked at you. You were pretty sure he was just waiting for smoke to start spilling from your ears.

You recognize something profound when you hear it. And, although you desperately want to keep up with the conversation, you just can't quite grasp what he meant. "Um, huh?"

"Yeah, you don't organize your grocery stores by recipes."

"No, I heard you, I just don't understand," you manage. And almost immediately, the light comes on.

A recipe uses a whole lot of different flavors to come together to form a new flavor combination. Sometimes, we can create a whole new flavor by changing just one of the ingredients. Organizing a grocery store by recipes means you would have to stock one product in thousands of places across the store – at least once for each time it occurs in a recipe.

You would have to know at least one recipe that calls for the ingredient that you want in order to ever find it. Then the insight starts to hit you. If you organized a

[8] Another excerpt from the soon-to-be-published *Choose Your Own Data Modeling Adventure* by Chris Allport

[9] This analogy is attributed to Dave Lombardi.

store that way, it would have to be HUGE. There are so many recipes. So, even though you use ground beef for meatballs and for tacos, it would have to be in two places. A third for hamburgers! And that means multiple freezer cases. How could you ever make a recipe that they didn't stock for in the store!?

Instead, ingredients are organized in a more logical way. Spices are usually found together – except sometimes those spices used only for seafood are found in that section. Meats are in a refrigerated case in one place in the store. Bread has a single place (maybe two if the store has its own bakery). And so on.

Building a data model is very much like organizing a grocery store. You don't duplicate data because you tend to use it in multiple places. You organize it logically and you visit that common place many times if necessary.

Why Should I Care?

Conclusion

If you live by principles, everything else is a derivative. Principles apply to any use case and they are still true whether you are doing code generation, specification construction, or performing validation of an implementation. A data architecture is a set of principles. Just like there isn't a linear flow to the process of modeling and documentation, there isn't a linear flow to the use of the model.

Note – like this paper, a data model can be built with the intention that there doesn't have to be a linear flow. One can leverage the data model at any point in the acquisition process, for any purpose. One just needs to know where to start.

Why Should I Care?

About the Author(s)

Gordon Hunt

Gordon Hunt is a co-founder of Skayl. His focus is on system of systems integration and semantic data architectures which enable increased systems flexibility, interoperability, and cyber security. Gordon's experience in building real systems with current and emerging infrastructure technologies is extensive. His technical expertise spans embedded systems, distributed real-time systems, data modeling and data science, system architecture, and robotics & controls. He is a recognized expert in industry on Open Architecture and data-centric systems and is Skayl's primary consultant for distributed real-time and service-oriented architectures and implementations. As a regular author and presenter, he speaks frequently on modern combat-system and command and control architectures and is an active member in numerous professional communities.

Chris Allport

Chris Allport is a co-founder of Skayl. Allport is a specialist in realizing concepts into advanced prototypes. He has a proven track record for creating and developing innovative & disruptive software products and is familiar with numerous aspects of unmanned vehicle industry: from standards to hands-on development. Throughout his professional career, Allport has been called upon to lend his expertise to myriad integration activities. In earlier years, these efforts were met with simple, point solutions, intended to solve the problem at hand. However, as the years have progressed and challenges more sophisticated, he has had to develop more sophisticated solutions to a variety of integration challenges. Recent projects are a sort of culmination of systems-of-systems integration.

Sonya Hand

Sonya is the Director of Marketing & Strategy at Skayl. Sonya has broad experience in the technology and business world. As an IT consultant with PricewaterhouseCoopers her technical knowledge, business savvy and strong communication skills helped establish the cadence of her career, as she became the fastest advancing executive in the company to-date. Upon leaving the PwC Technology Industry Team for the Mid-Atlantic region, Sonya established her own business consulting company, allowing her to operate with increased agility while continuing to serve and interact with companies the world over. Sonya has managed mass systems integration projects for customers including GE Information Services, WorldComm MCI, and AARP. Sonya is an expert strategist and technologist who is keenly adept at understanding and translating the business case for technology planning, implementation and application.

About The Open Group FACE™ Consortium

The Open Group Future Airborne Capability Environment (FACE™) Consortium, was formed as a government and industry partnership to define an open avionics environment for all military airborne platform types. Today, it is an aviation-focused professional group made up of industry suppliers, customers, academia, and users. The FACE Consortium provides a vendor-neutral forum for industry and government to work together to develop and consolidate the open standards, best practices, guidance documents, and business strategy necessary for acquisition of affordable software systems that promote innovation and rapid integration of portable capabilities across global defense programs.

Further information on FACE Consortium is available at www.opengroup.org/face.

Why Should I Care?

About The Open Group

The Open Group is a global consortium that enables the achievement of business objectives through IT standards. With more than 500 member organizations, The Open Group has a diverse membership that spans all sectors of the IT community – customers, systems and solutions suppliers, tool vendors, integrators, and consultants, as well as academics and researchers – to:

- Capture, understand, and address current and emerging requirements, and establish policies and share best practices
- Facilitate interoperability, develop consensus, and evolve and integrate specifications and open source technologies
- Offer a comprehensive set of services to enhance the operational efficiency of consortia
- Operate the industry's premier certification service

Further information on The Open Group is available at www.opengroup.org.