Disambiguate Data with Documentation

Context is the Key to Scaling Integration Complexity

Gordon Hunt, TRG Systems, gordon@trgsystems.com
Chris Allport, New Spin Robotics, chris@newspinrobotics.com

Abstract

Systems-of-systems integration is one of the most significant challenges facing the current generation of systems engineers. With the development of each new system, there is a combinatorial expansion in the integration effort. Unless the current trend is changed, the projected integration cost of these disparate systems could easily exceed the cost of the systems themselves.

It is necessary to disrupt this trend. By leveraging the power of existing data model standards, it is possible to document data so rigorously that even a computer can understand it. This process does not rely on sophisticated, next-generation, artificial intelligence algorithms: it relies on basic mathematics clearly based in the rules of set theory.

This is different than current system integration techniques that document and report data through defined message and protocol interfaces. Interface Control Documents (ICDs) – where each message in and out of a system is detailed and documented – capture the syntax and structure, semantics, and behavior in prose and diagrams. It isn't that the context and semantics are not understood, it's that that information isn't readily machine leverageable.

Using the techniques described in this paper, it is possible to reduce much of the interface and data integration process to a formal documentation exercise. When the interfaces are clearly documented, it is no longer necessary to have teams of engineers (or attorneys) arguing over the interpretation of an interface control document or the cost of changing it.

Applied properly, these techniques will substantially reduce integration costs and open up new pathways for communication between systems never intended to interoperate.

K**ey words:** integration, data model, documentation, context

Introduction

Integration complexity is a natural consequence of bringing together disparate systems designed by different teams. In some cases, these systems are intended to work together and are designed with other components in mind. When this happens, development teams may even make design choices with integration in mind. In other cases, systems are designed in isolation and teams share little (if any) data. After development is complete, the teams bring their myriad systems together only to find that they misunderstood the ad hoc integration documentation. Once the principal discrepancies relating to connectivity and choreography are worked out (i.e. basic communication is established), there are still several more levels of integration *on the data itself* that must occur.

The need to clearly document logical attributes of the data is well understood particularly with respect to units of measure and frames of reference.

Values must be represented by known units of measure. It is necessary to ensure that data are represented in the same units since it is not terribly meaningful to compare a Celsius value directly to a Fahrenheit value without conversion. Although individual systems may use their own unit selections, these values must be transformed into a common unit for comparison.

However, units themselves are not sufficient. It is not meaningful to attempt to integrate the body temperature of a human with the block temperature of an engine.

Values must also have clearly documented reference frames.

In 2012, the University of Canberra participated in the Outback Challenge – a search and rescue competition for autonomous unmanned aerial systems (UAS). After locating the target, the team reported the position to the judges for approval to drop a rescue package. The reported position was nearly 50 meters from the known position of the target and the Canberra team was directed to return their aircraft to base. After the competition concluded, subsequent analysis of the data showed that the Canberra team was actually within 10 meters. As it turns out, the UAS was using a different GPS reference frame than the judges used when they logged the position of the target. Although this error cost the team the competition (they returned to win in 2014), it illustrates clearly the need to ensure the reference frames are aligned just like the units. (Tridgell, 2012)

However, like before, units of measure *and* reference frame are not sufficient. Both heading and magnetic variance are referenced to true north (both may even be expressed in radians), but these are distinct attributes meaning very different things.

There are additional logical constructs that must be identified in order for successful integration to occur, but in the end, all will fall short because they are missing the most important part. What is most lacking is rigorous, unambiguous documentation of the *context* of the data.

Why Context?

Consider the following passage taken from a 1972 study by Bransford and Johnson:

> *The procedure is actually quite simple. First you arrange things into different groups. Of course, one pile may be sufficient depending on how much there is to do. If you have to go somewhere else due to lack of facilities that is the next step, otherwise you are pretty well set. It is important not to overdo things. That is, it is better to do too few things at once than too many. In the short run this may not seem important but complications can easily arise. A mistake can be expensive as well. At first the whole procedure will seem complicated.*
>
> *Soon, however, it will become just another facet of life. It is difficult to foresee any end to the necessity for this task in the immediate future, but then one never can tell, After the procedure is completed one arranges the materials into different groups again. Then they can be put into their appropriate places. Eventually they will be used once more and the whole cycle will then have to be repeated. However, that is part of life. (Bransford & Johnson, 1972)*

What are these two paragraphs talking about? The sentences are reasonably well constructed, the author seems to know what they are talking about, but the content is nonsensical at best. Can you recall any of the details from the passage? Can you put those details in any order?

Go back and reread the passage again, only this time, know that this passage is describing the process of doing laundry. Not only does this make more sense, but the details can be recalled more easily and even fit together in a reasonable sequence.

This same concept applies to data. Consider the message attributes in the following example:

```
interface Message {
    string assetID;
    long controllerID;
    PositionType position;
}
```

Figure 1 - Ambiguous Interface Definition

To what, exactly, do the attributes in this message correspond?

- Is this the controller reporting its currently controlled asset?
- Or is this the asset reporting the controller currently tasking it?
- Maybe it's the controller requesting control of a certain asset?
- Could it be an agent authorizing a controller control of a certain asset?
- Perhaps it's a conditional message that specifies the location where a controller must release an asset? Or maybe take control of it?

The message does not clearly indicate which of these questions the message addresses. The units of measure and reference frame do not provide any insight into these questions. Adding a better name to the interface such as "ControllerAssetReleaseCommand" adds some clues to interpreting the data, but many questions still exist.

When the interface documentation is adequate, it *will* capture this information, but at this point, the state of the art requires a human-in-the-loop to interpret that information. This is often written in ambiguous (or slightly imprecise, at best) prose.

Picking on Prose

Consider the following excerpt from an ICD: "send position updates more than the controller." What does this mean?

Clearly the author meant that position updates should be sent more frequently than controller updates. Or maybe this actually means to send position updates more frequently than the controller sends updates. When this instruction was written, the author had a very clear understanding of the intended meaning. In fact, the author's intent was so obvious (to the author), the familiarity bias may have prevented other interpretations from being discerned.

Even well-written text is subject to misinterpretation. Whether documentation is written collaboratively or written by a single author, there is still a problem with the interpretation of context: the determination of what is *actually* meant.

Thus even well-written documentation is not immune to the potential liability introduced by human interpretation. This is not intended as a justification for eliminating the human-in-the-loop: it merely emphasizes the need for an unambiguous method of documenting context.

Eliminating Amphiboly

Currently, integration is accomplished by commonality: human-inspired mediation and collaboration to create exchange patterns and structures for Systems-of-Systems integration. This is laborious and time consuming. What really are the integrators actually trying to accomplish? The humans and system architects are 'mediating' the interfaces. It isn't that mediation doesn't occur, it is just a question of where and when in the process it occurs.

What is needed is a technical solution that will allow systems engineers and developers to document context in an unambiguous manner. As this problem has already been solved many times, one needs look no further than structured, or programming, languages.

Computer languages allow developers to clearly document very complex ideas in a machine-usable manner. Once written, these complex structures can be interpreted by any computer equipped with the appropriate tooling (i.e. compiler). Once compiled (i.e. syntax is validated and transformed), the product of the "documentation" (source code) can moved from machine-to-machine and used again.

As with software development, specifying a structured language does not mean that everyone is required to write exactly the same thing, but they are required to follow the same rules for documentation.

Syntax for documenting context can be found in two existing standards: the Unmanned Aerial System Control Segment (UCS) Architecture and the Future Airborne Capabilities Environment (FACE) Technical Standard. Both standards use a similar notation. Not only is this notation clearly defined, it is machine-usable.

A machine-usable solution opens a whole host of possibilities for automating the integration process. With a rigorously documented message specifying all aspects of its attributes including context, it becomes possible to match attribute to attribute across standards. And, with all logical aspects of the message documented (e.g. units, frame of reference), it is possible to generate transforms so the data can be directly compared.

Conceptually Speaking…

Documenting is not terribly difficult, but it does require some advance preparation (i.e. a data model) along with an understanding of the subject matter.

Before a message attribute can be documented, it is first necessary to have some content and structure that can be used to document the attribute's syntax and semantics. Therefore, the first requirement is to have a data model. The data model must contain the definition for ***every concept*** used in the message set. This includes a definition for every unit, every reference frame, every measurement, every entity, and even every attribute.

This may seem like a cost prohibitive task. It would take a tremendous amount of resources (in both time and cost) to capture that volume of knowledge. Why is it reasonable to expect that any new or

existing program would invest in this amount of overhead? Furthermore, why would the customer, assuming a government customer, want to pay for this same overhead time and time again?

Constructing a domain specific data model can easily cost $20MM to develop. This figure is based on work that has already been completed and ***released to the public***. In other words, it is no longer necessary to capture the data from scratch. It may be necessary to document new information, but the existing body of knowledge provides a UAS domain specific baseline. And this data can be reused.

So what is in a data model?

As mentioned already, it must define all of the basic building blocks needed to build and describe entities, their attributes, their relationships to other entities, and their logical representations in an appropriate structure so it can be used in a repeatable, logical manner.

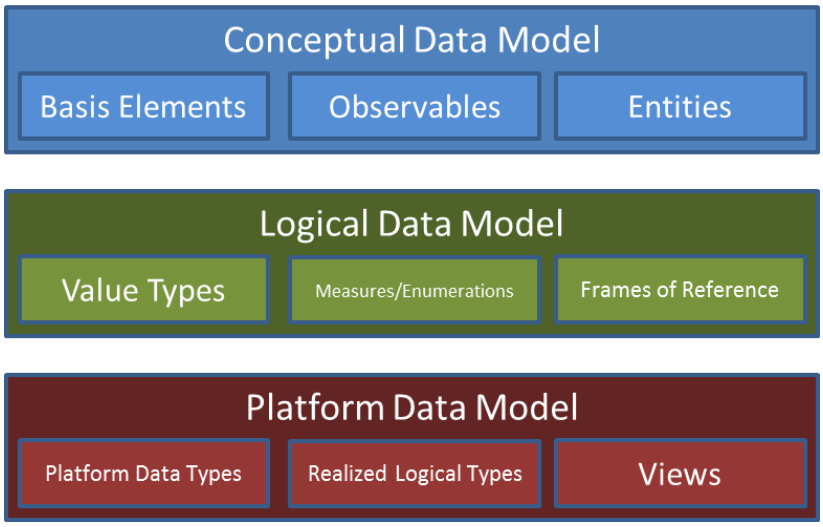

Figure 2 - Three Levels of Abstraction in a Data Model

As shown in Figure 2, the data model is represented at three different levels of abstraction: the conceptual, the logical, and the platform.

The Platform Data Model (PDM) level is the most concrete level of the model in which data is represented as it is in the message model – or as close to the message model as possible. It contains views which are composed of attributes. The attributes are the individual elements of each "message." They are called views in order to engender thinking of "database views" rather than an actual "message." Each attribute has a corresponding context definition. Views define are the blocks of data by which information arrives and leaves the system.

The Logical Data Model (LDM) defines the measures, value types, and other logical aspects of describing data. This is the level of documentation that most standards handle very well. After all, in many cases, this is where much of the work seems to be as it is non-trivial designing a mathematical transform between geographic projections. Fortunately, once defined, these same definitions (and their transforms) can be reused ad infinitum.

The Conceptual Data Model (CDM) is the most abstract level of the model. In addition to the building blocks of the domain (i.e. the pieces needed to describe everything in this domain), the CDM contains

the definitions of the entities. The entities are carefully modeled to document the composition of their elements and their relationships to other entities (and even to other relationships).

For example, a model of an aircraft can be constructed in the CDM. At this level, the aircraft does not have a "latitude and longitude," it merely has a position. Why? The reference frame, units, and representation are logical constructs and are elaborated in the LDM. At the CDM, position is just a concept and can be compared to other, equivalent concepts.

Therein lies the magic.

Historically, when one message refers to an aircraft's position in earth-centered-earth-fixed coordinates (X, Y, Z), the data does not resemble the geodetic position (latitude, longitude, altitude) in another message. Conceptually, however, they are the same thing.

The CDM provides the ability to compare concepts based on what they actually represent. This information is used to form the context. Thus, it is absolutely necessary to have a data model that has been well-informed by domain experts.

In order to properly document the ambiguous message proposed earlier (see Figure 1), it is necessary to define a data model that defines the named entities.

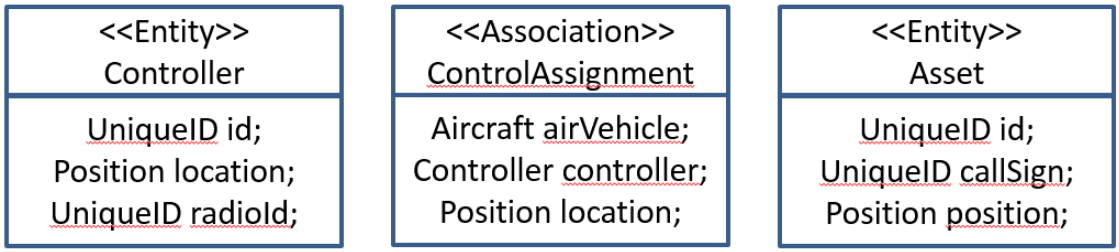

Figure 3 - Simple Data Model to Support Example

This simplified data model provides enough robustness to express many of the possible interpretations of the ambiguous message shown in Figure 1.

Building Context

Once the building blocks (observables, entities, and measures) are in place, it is possible to start documenting the context of each and every message attribute.

The first actual step in documenting the context is rigorous documentation of the message itself. The message must be represented in the PDM as a "platform view" element with each attribute represented as a child "characteristic" element.

```
<element xmi:type="platform:View"
       xmi:id="UNIQUE_GUID_of_Interface"
       name="Message">
       <characteristic xmi:type="platform:CharacteristicProjection"
                    xmi:id="UNIQUE_GUID_of_InterfaceAttribute_1"
                     path="->[ControlAssignment].asset.id"
                     rolename="assetID"
```

```
                    projectedCharacteristic="UNIQUE_GUID_Controller_Entity"/>
      <characteristic xmi:type="platform:CharacteristicProjection"
                    xmi:id="UNIQUE_GUID_of_InterfaceAttribute_2"
                    path=".id"
                    rolename="controllerID"
                    projectedCharacteristic="UNIQUE_GUID_Controller_Entity"
                    />
      <characteristic xmi:type="platform:CharacteristicProjection"
                    xmi:id="UNIQUE_GUID_of_InterfaceAttribute_3"
                    path="->[ControlAssignment].asset.position"
                    rolename="position"
                    projectedCharacteristic="UNIQUE_GUID_Controller_Entity"/>
</element>
```

Figure 4 - XML Documentation of Ambiguous Message

The characteristic element represents the message attributes. The context is encoded in the `projectedCharacteristic` and `path` attributes - both are required to fully express the context.

Continuing the example from Figure 1, the following four diagrams will show different interpretations of the message that will yield different context definitions.

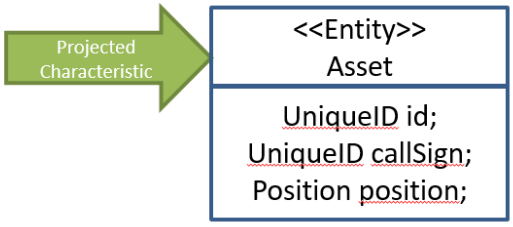

Figure 5 - Documenting Asset Id

This is the simplest context to specify. This is how a single entity is identified and is used to specify the context for the assetID field. In this case, the `projectedCharacteristic` refers to the Asset entity and the `path` simply refers to `id` attribute of the Asset entity.

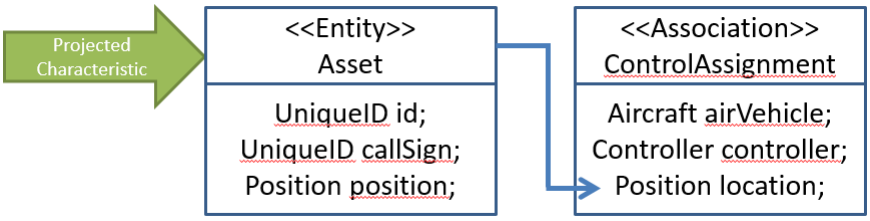

Figure 6 - Documenting the Position where Asset Control is Assigned

The example in Figure 6 is slightly more sophisticated. In this case, position refers to the position where the control of the asset is assigned. It seems appropriate to select ControlAssignment as the projected characteristic, and, in fact, that is the starting point.

Even if the documentation effort stopped there, it would represent a significant improvement in documentation of context. However, in this case, it is possible to do even better. To do so, consider the question, “position in the context of what?” The answer is Asset – the position where the ***asset’s***

control is assigned. Hence, asset is the projected characteristic and the path is "Asset → ControlAssignment.position".

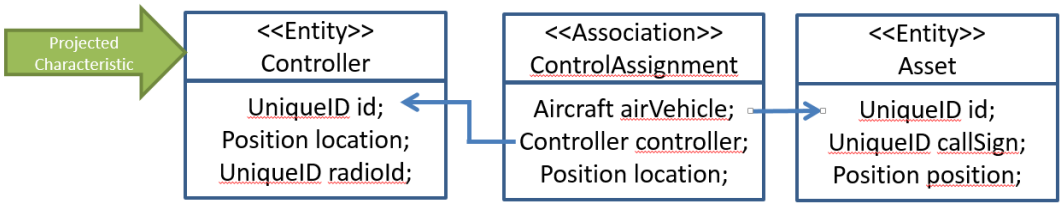

Figure 7 - Documenting the ID of the Asset that the Controller is Releasing

Unlike in previous examples where the relationships were simply navigating attributes, Figure 7 shows that it is possible to navigate the "other direction" to associated entities. The ability to navigate associations actually provides the developer with added ability to express meaningful context. The context diagram shown in Figure 7 represents "the asset that will be released by the controller."

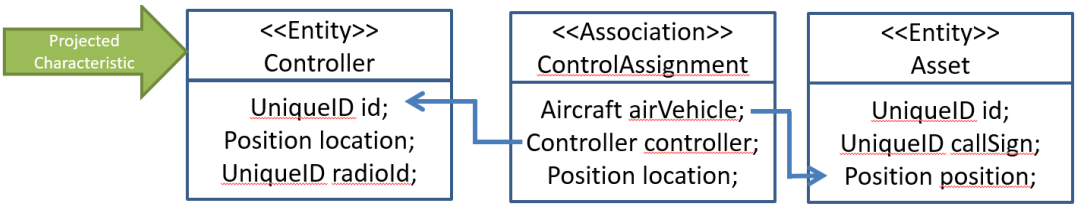

Figure 8 - The Position of the Asset that will be Released by the Controller

This context specifies the position of the asset that will be released by the controller. Once again, start with Asset.position and continue to build the context by asking "in the context of what?" This attribute is not simply the position of the Asset, it is the position of the asset related to the current controller. This relationship can be resolved using the ControlAssignment association.

Model → Document → Connect

Beyond supporting development of related industry standards, the authors' work on this topic to date has focused on the first two pieces of the ultimate solution: model and document. More specifically, a tremendous amount of effort has been directed toward developing tools that optimize the documentation workflow to allow the user be more accurate and efficient.

A shared data model was created by using the contents of the UCS Data Model. Though far from exhaustive, the shared data model leverages all of the subject matter expertise from UCS into the documentation format described above. This shared data model provides a common set of entities to which all messages can be referenced. When the context of different message attributes resolve to the same element in the shared data model, the two message attributes refer to the same thing. Then, by applying definitions from the logical data model, the first message attribute can be converted to the other. Since the context is machine-usable, this entire process can be automated.

Future work will consist of the next phase of the workflow: connect. Imagine a data model driven infrastructure in which the system integration can be performed by loading a documentation file. This will practically eliminate the need for costly, time intensive integrations. Instead of wrestling with precise wording in an interface control document (still subject to interpretation), systems engineers and

developers can spend a comparatively small amount of time documenting their interfaces as described in this paper.

The process of documentation effectively becomes the integration!

Conclusion

A data model isn't a luxury, it is a necessity. In fact, it has always been a necessity. The questions of structure, meaning, and behavior have been routinely answered by engineers integrating systems. This information isn't new. The need for the information isn't new.

What is new is how this information can be documented and leveraged for scaling the complexity of integration. There have been many standards, tools, and approaches developed for capturing a system's integration details. This paper has briefly introduced both a data architecture standard and a methodology that a tool can exploit to capture the data. Again, what is being documented is the full definition of the data: syntax and semantics. Documenting behavior comes next.

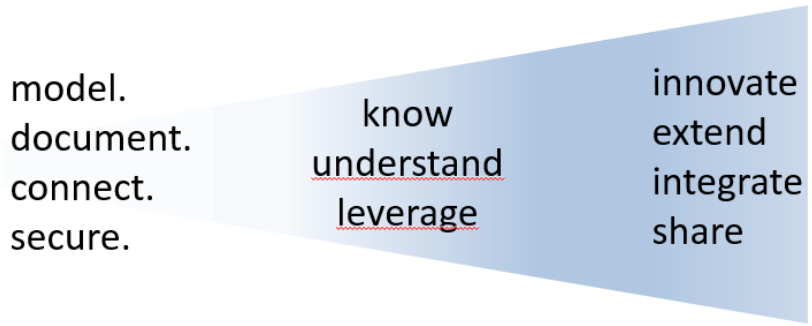

Figure 9 - What We Do, What We Need, and What We Want

Documentation, modeling, connecting, and securing is what we DO. That is, it's the mechanical aspect of getting to what we care about. The documentation helps an integrator to know, understand and leverage the data that is in their system. This is needed and necessary to get to what we really care about: innovating uses of the data, extending capabilities, integrating systems at scale, and sharing information with other systems. These things drive the need to get a handle on the complexity of scaling integration.

It is important to realize that scalability issues aren't completely new. In the world of data-at-rest and big data analytics, we are leveraging contextual models and semantics to process, discover, and correlate information in new and interesting ways. However, this scalability has been lacking for documenting context of data-in-motion.

Much in the same why early database structures transformed the world of information management, these contextual models and structures for data-in-motion will transform the world of systems integration. It is going to be an exciting development.

About the Authors

Gordon Hunt is the Principal and founder of TRG Systems. 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 TRG'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.

Current industry leadership includes serving as an appointed member of the Advisory Board of the Open Group's Future Airborne Capability Environment (FACE) architecture and SAE's UAS Control Segment (UCS) executive board and technical leadership. Gordon also supports many other interoperability military and commercial interoperability initiatives and industrial automation standardization efforts.

Gordon is a Captain in the US Navy (Reserves), a qualified Naval Engineering Duty Officer, and is currently supporting NAVSEA on cyber related concerns for naval combat systems.

Chris Allport wants to live in a world where obeying the law of gravity is optional, video games improve mental acuity, and spaghetti is health food.

As a software engineer with over 20 years of experience, he has lent his talents to developing new unmanned air systems, creating innovative software products, contributing to industry standards, and looking for new ways of solving existing problems where conventional methods have failed to work.

When he's not writing software and serving his team, you can find him enjoying his family, blogging about life, crafting home brew, and learning to ride a unicycle (a career in the circus is not anticipated).

His latest project —Modeling Wizard —hit the shelves in October 2015 and is currently available.

Learn about connecting the impossible at http://www.newspinrobotics.com.