Domain Specific Data Models (DSDMs): A Practitioner's Approach

Employing DSDMs in Support of a Model-based Modular Open Systems Approach (MMOSA)

The Open Group FACE™ Army TIM Paper by:

Sean Mulholland
TES-SAVi, Co-founder, CEO, & FACE SME

Bill Tanner
TES-SAVi, FACE Data Model SME

September, 2021

Table of Contents

Executive Summary

Domain Specific Data Models (DSDMs) have recently been employed with varying degrees of success. This paper explores DSDMs by building to a more formal view of them, what they are, how and when they should be employed, what a DSDM should contain, and approaches on how to leverage DSDMs to build easily integrable components supporting a Model-based Modular Open Systems Approach (MMOSA). Some best practices for employing and leveraging DSDMs are described with the intent of providing directly applicable methods for use in MMOSA systems.

Domains Specific Data Models

Overview

Domain Specific Data Models (DSDMs) have recently been leveraged with varying degrees of success as it is difficult to define what is a DSDM and how best to use them to "build up" a data model for a system. This paper explores what DSDMs are, how and when they should be employed, what a DSDM should contain, and approaches on how to leverage DSDMs to build easily integrable components supporting a Model-based Modular Open Systems Approach (MMOSA).

What is a Domain Specific Data Model?

For the purposes of this paper, we will need to define the meaning of Domain-Specific Data Model (DSDM). We assert the following definition:

> *a formal representation of data within a specific area of knowledge, bound by a particular defined category, at a level of abstraction and detail sufficient to enable specification and exchange for a particular use.*

How we arrive at this definition is important in that it is key to understand what a DSDM really is (or what it ought to be). Significant detail therefore is provided in the following section.

Understanding Domains

Within systems and software modeling circles, the term "domain" has been used in many different contexts, very often without any degree of specificity. That is to say that "domain" is a good term to use when speaking in generalities; however, if we want to get at the meat of what this term means and be of benefit to the reader and our modeling community, then we need to get more specific than what is required for the common elevator pitch.

Related to real-time embedded system modeling and prior to the advent of UML, the term "domain" was used to mean "subject matter". This term was preferred when describing the different knowledge areas (or areas of concern) typically represented in a system. A few recurring domains were the *application domain* (e.g., disk/tape storage and retrieval), the *user interface domain*, and the *process input/output domain*.

This traditional perspective leads us to the following definition:

domain*: An area of knowledge or subject matter bounded by some definition whereby some things are in the domain, and some things are not.*

We can take the definition of "specific" straight from Webster [7] to explain this term – both entries are equally appropriate for our discussions: *a: constituting or falling into a specifiable category, b: sharing or being those properties of something that allow it to be referred to a particular category.*

If we simply utilize Websters dictionary we can combine the definition of *specific* with our definition of *domain* to achieve the following:

domain-specific: *Constituting or falling into a specific area of knowledge, or subject matter bounded by a particularly defined category whereby some things belong in the defined category, and some do not.*

We can already see from our current definition of *domain-specific*, that a domain-specific data model will likely contain (if properly specified) only those things that belong to *our*[1] particularly defined and specific area of knowledge.

As with the terms domain and architecture, the term data model has been used for a very long time. A good general definition of "model" for our purposes is:

model: *any representation of something — at some level of abstraction and in some medium — the purpose of which is to specify some level of detail to either retain for one's own purposes, or to convey* to others.

The prefix "data" narrows the definition of model and limits it to one of the three accepted aspects used in the description of our systems, especially those of a real-time and embedded nature: data, control, and processing. Note that, in order to convey this representation to someone else (or even ourselves over a period of time), we need some form of conveyance, or exchange. This is where a more formal representation is necessary.

Now that we have informally defined the terms *domain*, *specific,* and *data model*, and have introduced this notion of formality, let's look at an informal definition of domain-specific data model before getting to the formal definition from the Future Airborne Capability Environment (FACE) Data Architecture:

domain-specific data model: *a formal representation of data within a specific area of knowledge, bound by a particular defined category, at a level of abstraction and detail sufficient to enable specification and exchange for a particular use.*

FACE™ Usage

We are now going to discuss a formal definition of a domain-specific data model, more accurately a FACE DSDM. Let's look at the definition in the FACE Technical Standard, Edition 3.0 [3]. From section "L Glossary" of the Standard:

Domain-Specific Data Model: *A Data Model designed to the FACE Data Architecture Requirements. It captures domain-specific semantics.* Here, also, is the formal definition of data model from the same Standard Edition:

Data Model: *An abstraction that describes real-world elements, their properties, and their relationships in order to establish a common understanding for communication between components.*

Alternate Definition of DSDM in the Standard

Someone familiar with the FACE Technical Standard, Edition 3.0 would likely point you to section 3.9.3.3 or FACE Technical Standard, Edition 3.1 section 4.9.3.3 which have this definition of DSDM:

[1] One would not necessarily think of the end of a screwdriver as a hammer, but if the abstraction within our domain is more akin to "something used to push a thumbtack into a hard wall", an example may well be a screwdriver, or even the heel of a shoe.

Domain Specific Data Models (DSDMs): A Practitioner's Approach

A Domain-Specific Data Model (DSDM) is a data model designed to the FACE Data Architecture requirements. It captures domain-specific semantics and generally does not contain Unit of Portability (UoP) Models.

The term "semantics" is used many times within the Standard, and in different subject areas/sections within the document. However, it is not defined in the glossary nor solely within a specific section. Further clarification is thus required.

Although a complete description of semantics is beyond the scope of this document, as is the differentiation between the FACE Conceptual, Logical, and Platform Data Model sub-types, the word "semantics" (in the broad sense as used in the Standard) connotes the concepts or meaning behind, or exemplified by, the data being modeled for the particular domain[2].

Common Understanding

An important tenet of the FACE Data Architecture is that the content of messages be defined in terms of their value types (i.e., Boolean, integer, float, etc.), units (feet versus meters), and frame of reference[3] (ECEF vs. WGS-84) to mention a few. What is just as crucial is that the FACE Data Architecture allows message content to be defined in terms of their "semantics". This means that a developer or integrator — looking at the definition of a message or a particular set of message data — must have access to the accompanying Domain-Specific Data Model in order to trace from the message data to the specified *context* and *meaning* of the data.

To clarify this by way of example, imagine that we have a message parameter that references a complete definition of position in terms of latitude, longitude, and height above an ellipsoid. All the details are known because the FACE Logical Model requires us to specify this completely.

What is still required? The context and meaning of the position within the domain being modeled. Several questions can be asked in order to identify the context and meaning of position. Is this the position of an aircraft, of a hostile track? Is it the position of an icon on the screen? The semantics define the meaning and context of the data, and most importantly prevent misunderstanding and ambiguity in the message data [1].

FACE Data Architecture

Both definitions of DSDM in the Standard make reference to the FACE Data Architecture. The FACE Data Architecture requirements are detailed in section 3.9.4 and, also, in the other cited sections that are both within and *outside*[4] the FACE Technical Standard.

Besides the more qualitative definition of what a FACE DSDM is, one thing is for certain: any FACE DSDM *must* adhere to the FACE Data Architecture requirements. This means that it must use a FACE conformant Shared Data Model (SDM) and must pass conformance by obtaining passing results as reported by the FACE

[2] More specifically, one may say that the semantics of the data can be considered the representation of the required knowledge according to its intended use.

[3] In the FACE Data Architecture, this is called a Measurement System.

[4] One must be careful to review and understand the many constraints specified in the Shared Data Model Governance Plan [4].

Conformance Test Suite (CTS). These are essentially the minimum requirements for a FACE conformant DSDM. Thankfully, there are currently no subjective requirements that look into a DSDM to see if it actually is well-formed according to someone's idea of domain-specific or correct modeling practices[5].

Elements, Properties, and Relationships

The elements, properties, and the relationships between them are just the "things" that are in our domain, their characteristics, and the relationships between them. For example, in a real or virtual battlespace, we would imagine that there are "things" such as friendly aircraft, hostile forces, and weather.

Characteristics (also called "attributes" or "properties" in other circles) might include a tail number or a universal resource identifier for an aircraft. Both would be unique, individual characteristics. For a hostile force, we might characterize it by its nationality (or unknown), its last known position, etc. For weather, we might want to know the volume of space it covers, or maybe the ceiling if it is a cloud.

As for relationships (also called associations) we might want to know how the friendly aircraft is related to both the hostile forces and weather, either for safety purposes or for routing planning.

If the definitions in this section were rewritten to use the terminology of the FACE Data Model Language, the terms would be equated as follows: Element would be roughly an Entity, property would map to Characteristic Composition, and relationship to Association. There are other elements (model elements) that are required by the FACE Data Model Language[6]. It is not as important to delve into these at this time, as our primary goal is to gain a better high-level understanding of a DSDM.

The "real-world" modifier may be confusing to some readers. As such, this is discussed in a dedicated section **Error! Reference source not found.**.

Communication between Components

This "real-world" part of the DSDM description for data model (because it is used in the definition for any data model) is too restrictive for general use as elements describe abstract concepts and not just "real-world" concepts. Nonetheless, it is applicable to a FACE Data Model. The use of the term "real-world" in the definition of data model means that when we are creating a data model that is representative of the elements in our subject matter, we name them according to what they represent in the world that we are modeling — the "real" world or "actual" world — according to the perspective that we care about. We have already touched on this in the previous sections.

For some of us, the real world will concern very tangible things like aircraft, terrain, weather, each according to our perspective, or knowledge area. This does not mean that all real-world elements are tangible[7].

[5] Note that there are two optional constraints, "Single Observable" and "Entity Uniqueness", that, when individually enforced, can significantly add to the difficulty in modeling a DSDM (or USM for that matter). If both are required, modeling will likely be impossible in some cases. Careful consideration is required when using these constraints.
[6] This is the second time we used the term "Data Model Language". This is simply the "language" we use to specify a conformant FACE Data Model, as defined by the FACE Data Architecture. Just as one writes C according to the ANSI C standard, one writes FACE Data Models in accordance with the FACE Data Model Language defined by the FACE Data Architecture.
[7] See Chapter 5 Classes and Attributes, from Executable UML, A Foundation for Model-Driven Architecture as one example [2].

Perfectly valid elements also include roles, incidents, interactions, and specifications. We bring this up because we don't want the reader to be under the assumption that a DSDM contains only those things that we can touch with our hands or see with our eyes

The use of "common understanding" in the DSDM description for data model, however, is not strong enough for a FACE Data Model. The primary purpose of a FACE Data Model is to improve the specificity of Component (UoP) interface documentation, thereby eliminating ambiguity and those errors that are a result of misinterpretation of the data being used in the communication between components. Most everyone has seen some version of a V-chart[8] and knows how costly it is to find ambiguities and miscommunications the further one progresses through the system lifecycle.

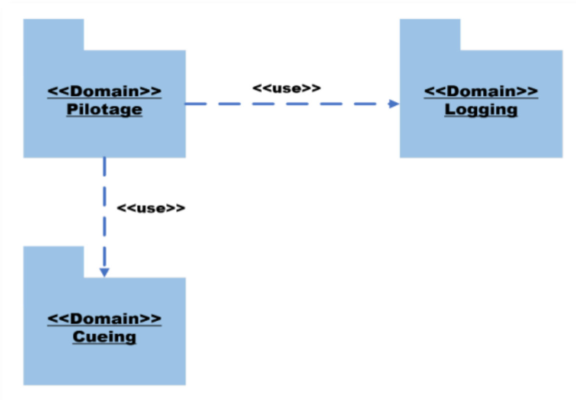


Figure 1 Example Domain Model with 3 Domains based on subject matter: Pilotage, Cueing, and Logging

Note that we are referring to data, not control or processing. Several attributes of the FACE Reference Architecture's Portable Component Segment (PCS) and Platform-Specific Services Segment (PSSS) components within the UoP model allow specification of communication style, message pattern, etc., and this information is very useful. Nothing in the FACE Data Architecture aims at a *fully specified* specification of control or processing in order to achieve lifecycle compatibility between components.[9]

DSDM Content

The content or the kinds of elements as defined by the FACE Data Architecture that comprise (or *should* comprise) a valid DSDM is going to be a topic of discussion over many years. It will be only from experience, and drudgery that the community will start to confirm exactly what should and should not be in a domain-specific data model for its various intended uses.

Although the definition in the FACE Technical Standard says that a DSDM doesn't "usually" contain UoP models, it is clear to these writers that inclusion of UoP models would be the *exception* not the rule. At least, it would be for those DSDMs that are currently being created.

In our view, the content within a DSDM is contingent on the domain, or subject matter being modeled, and the intent of its use[10]. In the two places where DSDM is defined within the Standard, each mentions semantics. To reiterate, this word is ill-defined and subject to much interpretation.

[8] If not, please see the introduction to [6] for several references.
[9] As of the Standard Edition 3.0, this is mainly limited to a few additions for lifecycle management, not choreography of components within the system.
[10] Within industry and government circles alike (besides technical aspects), there are procurement, IP, and other non-technical considerations.

Domain Specific Data Models (DSDMs): A Practitioner's Approach

Why a *Domain*-Specific Data Model?

Although pragmatism usually wins the day, especially when one does not have the luxury of unlimited time and schedule, it behooves us to think about the ideal when considering domain separation. Especially when the end-goal is to provide an environment where semantically unique domains can be purchased and integrated with other similarly-built artifacts.

We are not going to get our domain-specific data models right the first time, nor should we try[11]. In current practice, there will be (it has already been the case) that all the domains for a given usage are placed in the same model, viewed as one single "domain", such as the domain of "mission system" or "avionics" or "sensors". The authors are suggesting a slightly different approach, even when considering that 1) the FACE Data Model language was not designed specifically with multi-domain support in mind and 2) the model elements within the FACE Technical Standard 3.0 that do support this have been placed under the control of the FACE SDM CCB[12].

The suggestion advocated in this paper is to create domain-specific data models that are actually domain-specific, not application-specific. Domain-specific data models describe domains that are *not* functionally specific based on current function descriptions, or duplications of ICD/IDD message sets, but descriptive of the *essence* of the subject matter being modeled. It will be only through creating data models of actual subject matters that we will get true reuse and be able to take advantage of the work supporting model-to-model translation of platform independent (i.e., not platform-specific or platform-centric) models. This will likely be the only way in which the FACE Technical Standard matures beyond its current state of only generally covering and supporting the notion of domain specific data modeling.

A Few Words about the Integration Model (IM)

At the time of this writing, the Integration Model is not required to obtain conformance of one's Unit of Portability (UoP) or DSDM. Within the Integration Model, however, there are several model elements that an integrator can use, most notably being the View Transformation. It is a construct that allows for the semantic translation of data, from one specified View to another, as the data passes along a Transport Connection from one Unit of Conformance (UoC) to another.

Although in its infancy, this and similar constructs provide for the transformation of the point of view, or semantics of one domain, to the point of view, or semantics of another domain. This type of transformation specification, from one semantic representation to another, is crucial to separating the current assemblage of subject matter soup that plagues existing systems and thereby limits, reduces, or even eliminates opportunities for realizing the various "-ilities[13]".

[11] Our efforts have been significantly hampered by an attitude of "get it right or don't send it out" for data models. This can only hurt us in the long run as it prevents significant learning experience – the aforementioned drudgery.
[12] As anyone who has experience with this knows, especially in terms of adding content to the SDM, this can be an unrewarding, painful and often defeating process. Thankfully, one can most often find a doable "workaround".
[13] Maintainability, sustainability, extensibility, etc.

How and When DSDMs Should be Employed

As with all new technologies and processes, the "hows", "whats", and "whens" are based on many factors, not just project management related cost, schedule, and risk. Maturity of standards, processes, & tools, and knowledge and training among the engineers are all significant. This section therefore provides some general guidelines based on use cases. The FACE Data Architecture Interoperability Group (DIOG) Guidance Working Group[14] developed the following Use Cases[14]:

1. A **community of interest** wants to **share** agreed upon data model elements for their domain(s).
2. For **procurement** of UoCs that meet a **specification** defined in Conceptual & Logical DM and AbstractUoP.
3. **Integrator** needs to **connect UoPs** and creates a DSDM that UoC Suppliers should use.
4. UoC Supplier needs to **build a UoC** that meets a DSDM **specification**.
5. **Extending** core DM elements for some purpose (1 or 2) to further **define a domain** of interest.
6. UoP **Supplier** needs to **build FACE software** and doesn't want to start from scratch.

For this paper we will focus on two primary use cases for DSDMs that are applicable for the majority of the community concerned with MOSA[15] principles:

- Describing a domain of interest with intent for reuse and sharing[16]
- Describing capability interfaces with a high-level of specificity[17]

Quite naturally, the first use case is supportive of the second, but does not require it. However, in order to describe capability interfaces with the desired/required level of specificity, the domain of interest that describes the capability data must also be specified to such a degree.

Describing a Domain of Interest

DSDMs can/should be used to describe a domain of interest when the intent is reuse and sharing among a narrow or even broad community of interest. The specificity of the FACE Data Model Language lends itself to the task of describing with a great deal of specificity not only the concepts of the domain of interest, but their interrelationships, and even laying the foundation for the context of information exchanged between abstract and concrete components and their capabilities.

[14] The DIOG Guidance Composable Data Model document also contains useful information about the complexities of building data models in a layered way and getting them through conformance. At this point, the authors of this paper believe the only method that can be successful for managing SDM, DSDMs, and USMs is through third party tools such as AWESUM.
[15] To support enabling of technological innovation, incremental improvements, integration, and interoperability.[9]
[16] For example: designing severable modules, defining the interfaces between modules, and defining, standardizing, and describing data models.[8]
[17] Ibid.

Based on experience and significant participation in both government and industry projects throughout the last decade, it has become apparent that a set of correct, concise, and properly partitioned (as advocated above) single source of truth domain models is needed. Without this, and without a commitment by stakeholders to create, update, use and adhere to these models, our community will continue to waste valuable and unrecoverable time not only on fruitless arguments about "which terms are better" but also on recovering (if possible) from inconsistencies, misinterpretations and bad assumptions.

As was stated previously: we have to start describing domains, even if our descriptions and data model constructs are not "perfect" and our domains do not contain properly partitioned subject matter elements. Suggestions for modeling properly partitioned domains are provided below.

Describing Capability Interfaces

The FACE Technical Standard, Edition 3.x provides a very powerful Data Architecture (DA) designed to provide mechanisms to define interfaces with a high degree of specificity which is essential to procurement and validation of components. While the DA requires significant work to specify, define and refine interface definition, it increases the likelihood of successful component development, integration, and validation of complex systems due to the high level of specificity it requires[10]. While the FACE Data Architecture has many benefits toward portability and specificity, it does not prescribe how the interfaces should be defined. This is where the concept of Capability Interfaces is introduced.

Capability Interfaces are built utilizing the Capability Driven Architecture (CDA)[18]. CDA is a reference architecture which promotes software reuse by abstracting platform environment and device capabilities into common or standard interfaces. This methodology is supported by open platform tools and is backed by a rich history of safety critical aviation flight software systems[11]. This approach at developing Interfaces leveraging both the CDA and FACE Reference Architectures has seen significant success as documented in the several papers located at https://tes-savi.com/[19] . Capability Interfaces, when applied to domains, promote developing reusable interface definitions by abstracting the interfaces to be non-device, and non-application specific.

What Elements Should a DSDM Contain

The key to constructing DSDMs lies in understanding their expected use. When describing the concepts of a domain of interest, a complete Conceptual Data Model with a rich set of Entities and Associations would likely be all that is required. When greater specificity is required, especially due to the abstract nature of elements within the Conceptual Observation perspective, a fully realized Logical Data Model should be specified using Measurements from the SDM, or ones created that reflect the constraints of the given domain. Typically, this is as much detail as one would expect to be contained in a DSDM that is developed for documenting a domain of interest.

[18] The TES-SAVi AWESUM tool suite has a license for CDA, a patented technology, so that users of AWESUM can readily build Capability Interfaces with links to the FACE DA to produce FACE conformant Capability Interfaces.
[19] These publications are located on this website and require a free registration to access.

However, if the model is developed to specify a set of capability interfaces for a particular domain, and the specification must include not only the conceptual meaning of the leaf data elements but also the context, then it will be useful to also specify the set of Entities, Associations, and their relationships that provide this context. This can be done by specifying either Conceptual Queries or Logical Queries depending on the level of specificity required for the leaf data elements. Note that currently in the FACE Data Architecture, the only way to define groups of interfaces at the Conceptual and Logical levels is to employ Abstract UoPs. The Abstract UoP is defined as having Abstract Connections that are described by Conceptual or Logical Query Views.

Lastly, it can be tempting to specify interfaces at the Platform Data model level to ensure complete compatibility and replaceability of UoCs. The issue with requiring this level of detail is that it can overly constrain the UoC implementor who arguably is the Subject Matter Expert (SME) for the UoC. This is an area where Capability Interfaces can be employed to create general interfaces that support multiple UoCs while retaining the flexibility to support varied implementations.

Common Issues in Leveraging DSDMs

There are several common issues that practitioners should be aware of when using DSDMs in their current state of maturity within the context of a Model-based Modular Open Systems Approach (MMOSA). These are grouped into categories:

1. DSDM element mismatch to Unit of Portability Supplied Model (USM) needs
2. Conflicting DSDM elements (XMIIDS or Names) with other DSDM or USM elements
3. Different versions of FACE Technical Standard Editions or SDM versions
4. Configuration management issues of SDMs, DSDMs, and USMs and the need to update, merge, review data models
5. Security levels of the Data Model (DM) files: Affects the accessibility of the supporting DM files
6. How to validate multiple FACE (XMI) files when the FACE Conformance Test Suite (CTS) only supports one file[20]

Modeling Properly Partitioned Domains

Below are a few suggestions for the incorporation of properly partitioned domains into your own DSDMs, or into your USM that is reliant on DSDMs. Note that some of these suggestions may be limited by your ability to make modifications to an existing DSDM or make use of it in novel ways.

[20] The FACE DIOG has researched these issues and is working some mitigation efforts. The expectation is that CTS will continue to support only one file for conformance testing since many conflicts require human intervention to manage and perform resolution.

Organization

The authors have had success in organizing large multi-domain models into packages that are named and described according to the actual domains and sub-domains that are being modeled. Even though a very large conceptual model may contain elements of varying and even distinct subject matters, these organizational approaches can be used, even when the content is unchangeable, as is the case with Shared Data Models and certain DSDMs.

Many tools allow for the use of references to actual model elements. Practitioners can create packages with names that align more to their stakeholder's expectation, and also limited to only those elements used for messages specified by the USM.

For example, imagine that there is a DSDM that has thousands of elements for multiple domains, and they are packaged into folders named "A", "B", "C", and so on through to "Z". If the USM is only using a subset of the model elements, perhaps just "Track" and its related elements, and "Message" and its related elements, two packages, that contain these references, could be created as is shown in the figure below where we see a Unit of Portability Supplied Model (USM) referencing DSDM model elements and "re-organizing" those elements into a more "natural" semantic structure.

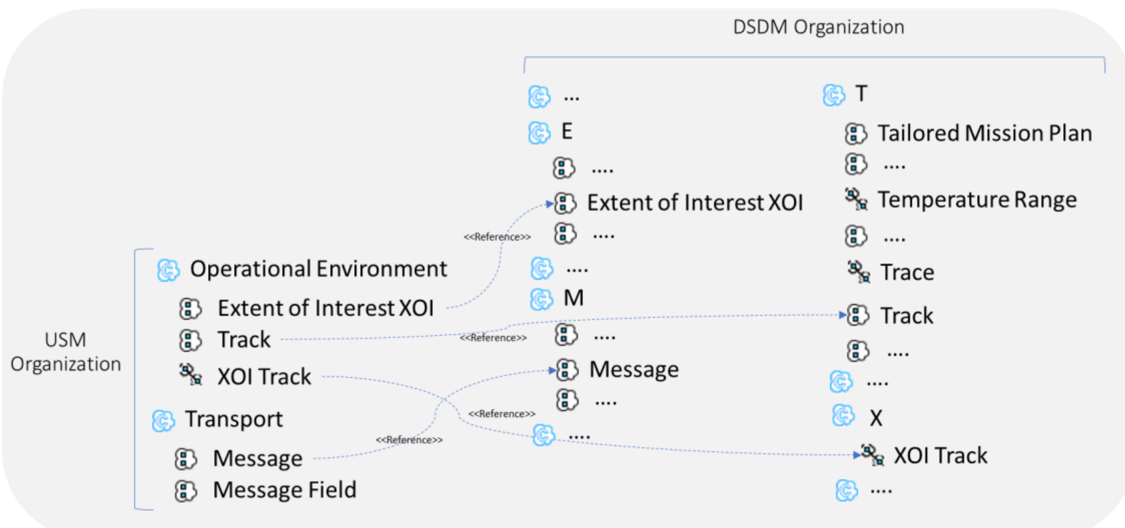
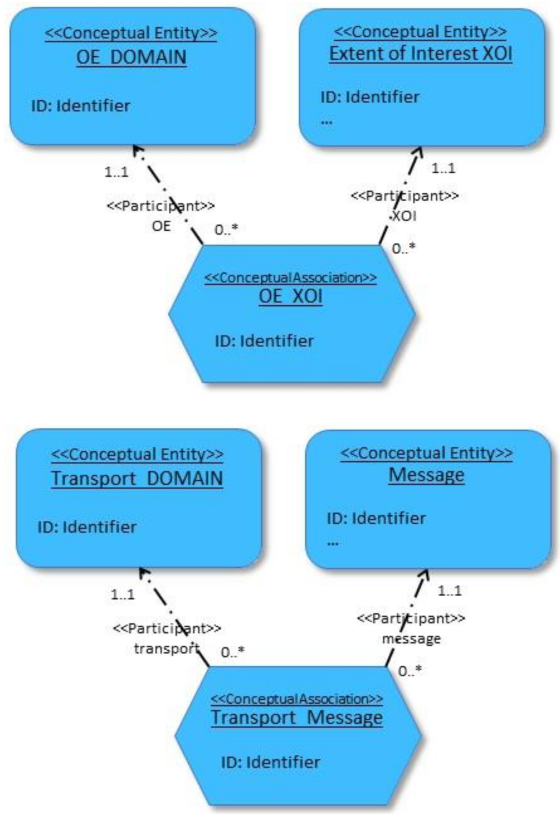


Figure 2 Example Organization of a USM leveraging DSDM model elements

Direct Identification of Domains

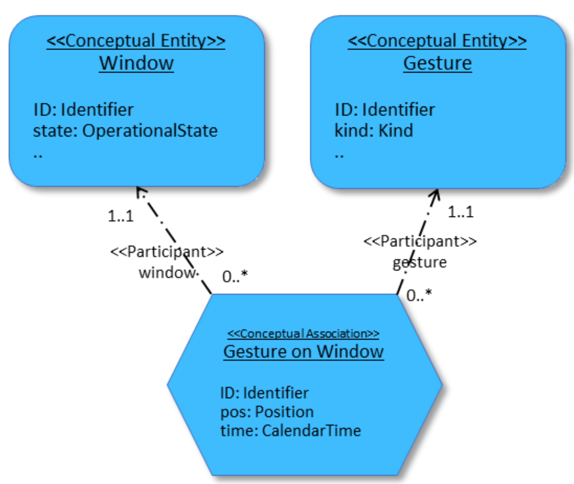
In order to directly identify the domain to which concepts belong, and in absence of any formally-defined Domain elements defined within a Shared Data Model, Associations can be used. This may be a bit more overhead than some would like, but with a single model that contains multiple domains (especially when the same name appears more than once), the effort is likely outweighed by the benefits. This is especially true when accompanying supporting processes are consistently used.

The examples below show an Association tying Extent of Interest XOI to the OE Domain and Message to the Transport Domain respectively.

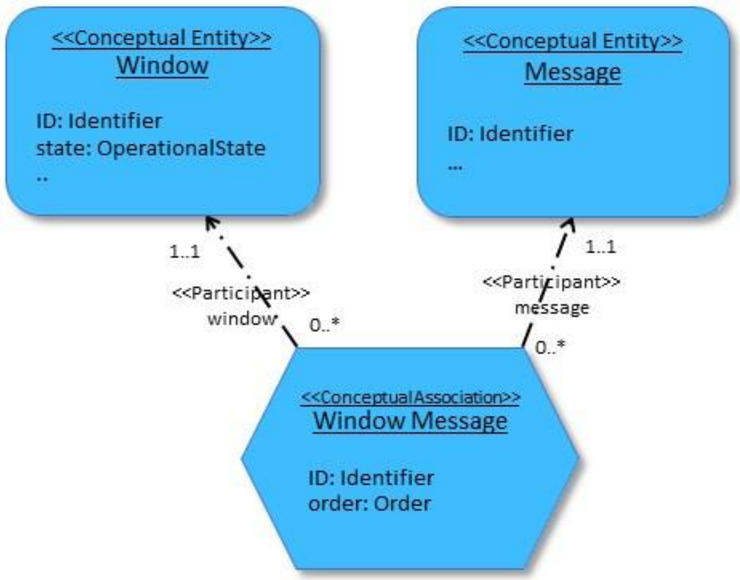


Mapping between Domains

There are times when a mapping should be modeled between elements in different domains or sub-domains. There are valid reasons to do the mappings when describing the relationship between the two elements. For example, one might want to associate Window and Gesture, which are theoretically in different domains:



In many cases, modeling of this mapping has to do with the desire to create a path between two disparate model elements because one would like to use them within the same Query (i.e., Join them):



Because Window and Message are semantically disjoint, they would never likely be part of the same domain-specific data model, although within the solution space, they are required to be.

Summary/Conclusion

The main intent of this paper is to provide a more formal view of DSDMs, understanding the use cases for DSDMs and to describe some of the best practices employed with leveraging DSDMs for use in MMOSA systems.

A secondary goal is to open the conversation to other details and aspects of the problem of creating DSDMs, leveraging DSDMs for USM development, and even going so far as to provoke discussion regarding domain separation and semantic mapping among disparate but dependent domains.

The authors look forward to feedback from our user base and community and working on follow-on papers required to assist in the realization of the full potential of our Model-based Modular Open Systems Approach (MMOSA) to systems and software engineering with the AWESUM® Tool Suite.

Quotes:

"Model with a purpose, Know your purpose before modeling"

"Models will never have the same level of detail as the system they are modeling"

"Modular systems are cool, Reusable systems are AWESUM!"

References

1. Mars Climate Orbiter, Mishap Investigation Board, Phase I Report, November 10, 1999.
2. Mellor, Stephen; Balcer Marc: Executable UML, A Foundation for Model-Driven Architecture. Addison-Wesley.
3. The Open Group (2017), FACE™ Technical Standard, Edition 3.0; retrieved from www.opengroup.org/library/c17c
4. "FACE Share Data Model Governance Plan, Edition 3.0", X1817US. https://www.opengroup.org/face/
5. "*AV-2 for FACE 3.0*", Open Group FACE Plato site
6. "Air Force's Continuous Integration and Continuous Development with the FACE™ Technical Standard, FACE Army TIM 2018", Huntsville AL. https://tes-savi.com/publications
7. https://www.merriam-webster.com
8. "*Modular Open Systems Approach (MOSA) Panel on Standards*", Zimmerman, Phil, Defense Standardization Program Workshop, July 12, 2018
9. "10 U.S. Code § 2446a - Requirement for modular open system approach in major defense acquisition programs; definitions", (4) The term "major system interface" part (b).
10. "Buyers, Suppliers, and Integrators, Oh My! FACE Army TIM 2020", Huntsville AL. https://tes-savi.com/publications
11. "Capability Driven Architecture an Approach to Rapid Platform Integration", Tucson AZ, Jan 2012. https://tes-savi.com/publications
12. "Composable Data Models (DSDMs)", Open Group FACE Plato site https://www.opengroup.us/face/diog/guidance/protected/documents.php?action=show&dcat=&gdid=22797

About the Author(s)

Sean P. Mulholland is a co-founder of Tucson Embedded Systems, Inc. Sean has a B.S in Computer Science and Systems Design from the University of Texas at San Antonio. Sean currently serves as TES CEO and President. Sean has 31 years of experience in software intensive system development, design, integration and testing, especially as it relates to mission critical and safety critical systems. Sean has designed and built several product lines that produced significant advancements in the areas of Geographic Information Systems, Military Ground Systems, Unmanned Ground Vehicles, Unmanned Aerial Vehicles, and Manned aircraft systems. Sean is a contributing author to the FACE™ Technical Reference Architecture and has been active serving as a key resource in FACE Data Architecture development and serves as co-lead in the Domain Interoperability Working Group (DIOG) Guidance Group. Sean's current work is focusing on the development of a process and supporting tool suite for optimizing the system development of safe and secure systems for military and commercial systems.

William G. Tanner is a contributing author to the FACE™ technical reference architecture and has been active in the FACE Domain Interoperability Working Group (DIOG). Bill is currently the lead data modeler for several TES, TES-SAVi, and Army projects across varying subject matters. Bill has a B.S. in Computer Science Engineering from Northern Arizona University and over 28 years of embedded software, embedded software application development, and modeling experience, half of which are in software engineering project and product management, the other half directly related to systems and software engineering and modeling using UML, xtUML, and FACE™ Data Architecture. Prior to joining TES in 2006, Bill worked for IBM as a software engineer in the disk storage system division, and Project Technology and Mentor Graphics as a data modeler and project manager for the division's UML modeling, verification, and code generation tools.

About The Open Group FACE™ Consortium

The Open Group Future Airborne Capability Environment Consortium, (the 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 the FACE Consortium is available at www.opengroup.org/face.

About The Open Group

The Open Group is a global consortium that enables the achievement of business objectives through technology standards. Our diverse membership of more than 800 organizations includes customers, systems and solutions suppliers, tools vendors, integrators, academics, and consultants across multiple industries.

The mission of The Open Group is to drive the creation of Boundaryless Information Flow™ achieved by:

- Working with customers to capture, understand, and address current and emerging requirements, establish policies, and share best practices
- Working with suppliers, consortia, and standards bodies to develop consensus and facilitate interoperability, to evolve and integrate specifications and open source technologies
- Offering a comprehensive set of services to enhance the operational efficiency of consortia
- Developing and operating the industry's premier certification service and encouraging procurement of certified products

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