MBSE without a Process-Based Data Architecture is just a random set of Characters

Robert K. Crain
MEI Technologies, NASA Johnson Space Center
17102 Chapel Park Way
Houston, TX 77059
281-483-0922
Robert.K.Crain@nasa.gov

Abstract— **This paper describes the importance of developing a Process-Based Data Architecture as part of pursuing a Model Based Systems Engineering Methodology. The purpose of this paper is to introduce the user to best practices in developing a Model Based Systems Engineering (MBSE) approach, focusing on the definition of the Data Architecture.**

TABLE OF CONTENTS

1. INTRODUCTION

MBSE without a process-based Data Architecture is just a set of random characters….. 'M'… 'B'… 'S'…'E'.

So, what is MBSE? Four letters that represents - Four words that are comprised of even more letters (Model, Based, Systems, Engineering) - which is represented as an acronym.

For us to understand what M-B-S-E is someone had to: define letters; arrange those letters into words; provide definitions for those words; combine the words together to provide a specific meaning; and finally, define the term acronym and how it is applied.

So, what is Mfano Mifumo ya Uhandisi ya Msingi (MMUM)??? The words Model Based Systems Engineering (MBSE) translated into Swahili. Because we have the underlying defined structure (letters, words, definitions) our words can be translated into other languages.

- **Modelo de la ingeniería de sistemas basados**
- **Modello base di sistemi di ingegneria**
- **نموذج هندسة النظم القائمة**
- **Μοντέλο που βασίζεται μηχανικής συστημάτων**
- **基於模型的系統工程**
- **Odelmay Asedbay Ystemssay Engineeringway**

All of these examples follow the same underlying "process" for creating an "architecture" to "model" words that have a "meaning".

A tool agnostic Process Based Data Architecture provides this same structure for implementing a Model Based Systems Engineering (MBSE) Methodology. It allows Models to be created in any "language" SysML, UML, Cradle, Core, DOORS, etc. and facilitates the transformation of the model from tool-to-tool and data integration mapping.

2. PROCESS-BASED DATA ARCHITECTURE

A Process-based Data Architecture provides an understanding of what information is needed to effectively execute the enterprise's business processes and provides a framework for effectively managing the enterprise's information environment. It provides a representation of data artifacts and data assets that classifies and defines all data entities, their attributes, and associations to facilitate knowledge of how data is produced, managed, and shared in different contexts of use. The Data Architecture provides identification, management, interoperability, and integration of information across business or organizational elements needed to support Product Data Life-cycle Management (PDLM) goals. It also ensures data needed by programs and projects (e.g., for milestones, reviews, mission operations, and anomalies or investigations, decisions, and outcomes) are identified and managed to provide traceability of data used in decision making. In the past, the primary methodology for accomplishing this process has been highly Document-centric. However, as systems get more complex and the amount of "Big Data" expands, conventional methodologies induce undo risk and are unable to provide the highest fidelity of data necessary to make the best decisions.

978-1-4799-1622-1/14/$31.00 ©2014 IEEE

3. MBSE FOUNDATION DEVELOPMENT

We understand the meaning of the letters MBSE because: somebody defined letters; arranged those letters into words; provided a definition of those words; combined a series of words to provide a specific meaning; and finally, defined the term acronym and how it is applied. If all of this upfront work did not occur, the letters MBSE would be somewhat meaningless and it would be extremely difficult to translate. These concepts are true for Model Based Systems Engineering development. If the upfront work (the Process-Based Data Architecture) is not properly executed to provide a road map for the SE model, the results could be random and require extensive rework to provide a cohesive, translatable result. The best part about the Data Architecture (DA) model is its static and tool agnostic. Generally, once the DA model is developed it is used repeatedly as a reference when developing the Application Architecture (tool) layer of the Enterprise Architecture.

The resulting "Ontology" is tool agnostic and represents standard relationship represented in a document-centric environment. (ex. Verifications verify Requirements which specify an Architecture used to achieve a Mission that is guided by Needs, Goals and Objectives). It is important to keep in mind that both Document-centric and Model-centric Systems Engineering both share a common process. The difference is in the methodology(ies) for implementing those processes.

4. BENEFITS OF DATA ARCHITECTURES

A well-defined data architecture will: facilitate standardization of definitions, ensuring stakeholder concerns are satisfied; discovery of document based process flaws; identification of processed based compliance audits; ensure data is interoperability and can be exchanged; provide cohesive integration of data across disparate tools; provide guidance for application/tool selection; aid in methodology development and process traceability; drastically reduce the time to develop application schema definition; identify Information Technology Application layer requirements for hosting applications and conducting federated search, query, auditing and reporting from the authoritative data source.

Additional Benefits of the Data Architecture

- Provides an understanding of what information is needed to effectively execute the enterprise's business processes and provides a framework for effectively managing the enterprise's information environment.
- Links information behavior (i.e., accessing, using, and sharing data), information management processes, and information support staff to other aspects of the enterprise.
- Provides a (process-based) representation of data artifacts and data assets that classifies and defines all data entities, their attributes, and associations to facilitate knowledge of how data is produced, managed, and shared in different contexts of use (viewpoints).
- Provides identification, management, interoperability, and integration of information across business or organizational elements needed to support program PDLM goals.
- Ensure data needed by programs and projects (e.g., for milestones, reviews, mission operations, and anomalies or investigations, decisions, and outcomes) are identified and managed to provide traceability of data used in decision making.

Following a logical 10 step process, which includes Data Architecture Definition, will ensure the desired MBSE results are achieved and consistent to provide these benefits. (See Figure 1 – Ten Step MBSE Development Process)

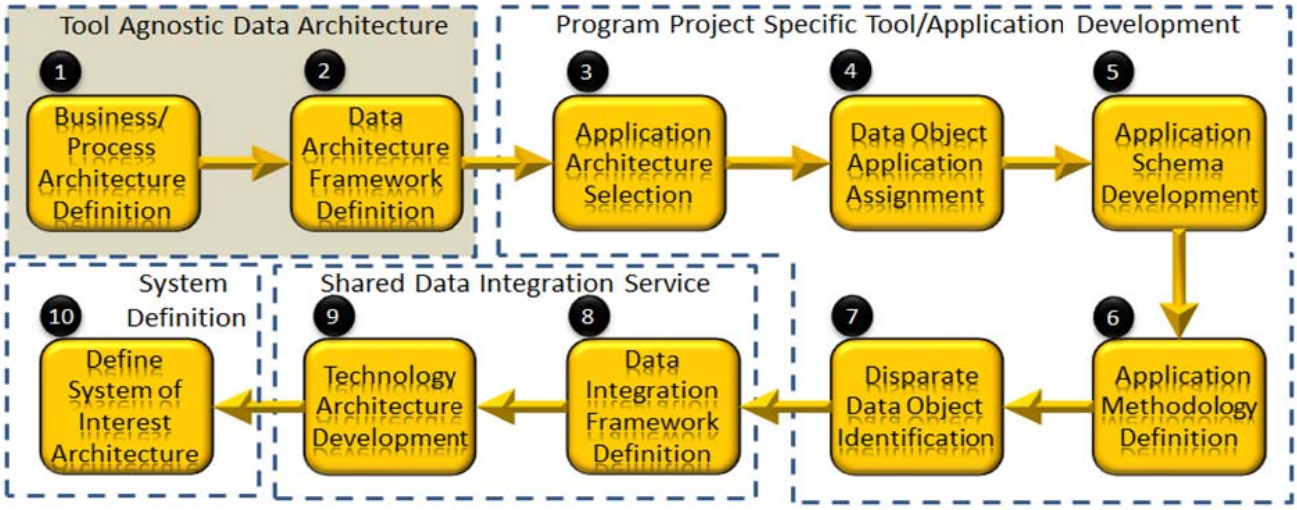

Figure 1 - Ten Step MBSE Development Process

5. TEN STEP MBSE DEVELOPMENT PROCESS

1- Process/Business Architecture Definition

The foundation to any Model Based Systems Engineering development is a well-defined and understood set of Business Rules/Processes. These processes provide the guidelines and road maps for implementing the model based approach. The good news is these processes should already be well established if you are currently using a Document Based Systems Engineering (DBSE) methodology. MBSE does not change the underlying defined and approved standard operating procedures (SOPs) and processes. What is changing is the methodology for implementing these processes. Therefore, in many cases, this step of the process has already been completed.

The position of the system-of-interest within the hierarchy of the overall architecture will determine the specific guiding Business Architecture rules for Systems Engineering and Project Management process requirements. The Program/Project could adopt established Corporate standards, tailored Corporate standards, tailored Program/Project standards, or decide to create new standards and processes, which meet or exceed the parent standard(s). For Programs/Projects these documents would typically be included as Applicable Documents in the Systems Engineering Management Plan (SEMP).

2- Data Architecture Framework Definition

Similar to the Process Architecture definition previously being defined many elements and viewpoints [1] of the Data Architecture may have already been defined from a Document standpoint. These Data Architecture elements are easily extractable from document based forms, templates, excel spreadsheets, etc. The main effort in this step is to extract the data objects, attributes and association to other data objects contained within the document based forms and templates. The next step is to ensure this document based artifacts map to and meet the intent of the Business Rules/Processes. In many cases document based systems can mask process deficiencies due to the vast amounts of disparate data spread across multiple documents. Additionally, data extracted from documents, rather than derived from the defined process, must be normalized. Normalization is the process of interrogating the Data Architecture and reducing or eliminating redundancies and dependencies. Process based data architectures should not require normalization, unless of course the process is redundant or duplicative. In this case the process itself might require normalization.

Based on the Program/Project Business Architecture definition a Data Architecture "Framework" (DAF) can be adopted, tailored or newly developed. The Data Architecture "Framework" identifies the Data Objects, attributes, and associations (relationships, links, correspondence, etc.) applicable to Program/Project. Existing DAFs can be tailored and expanded to add optional Program/Project specific attributes and/or associations. Deviations/tailoring to the common Data Architecture model should be documented and well understood so the data can evidentially be transformed for data exchange, integrated reports, audits, and metrics. Unique program/project specific additions should also be documented in the Program/Project's Data Architecture Dictionary. The resulting DAF should be documented or referenced in the Systems Engineering Management Plan (SEMP).

Data Architecture Data Object examples include Mission, Needs, Goals, Objectives (NGOs), Schedule, Work Breakdown Structures (WBS), Cost/Budget Breakdown Structures (CBS), Organizational (people, boards, panels) Breakdown Structures (OBS), Requirements, Functions, Operations, Risks, Analysis, Capabilities, Validation, Architectures, Products, etc. Domain specific subsystem areas such as software, power, thermal/cooling, structures and mechanisms, data systems, etc. are instantiations of the Architecture/Product Objects. (See example Figure 2 – Sample Data Object Diagram)

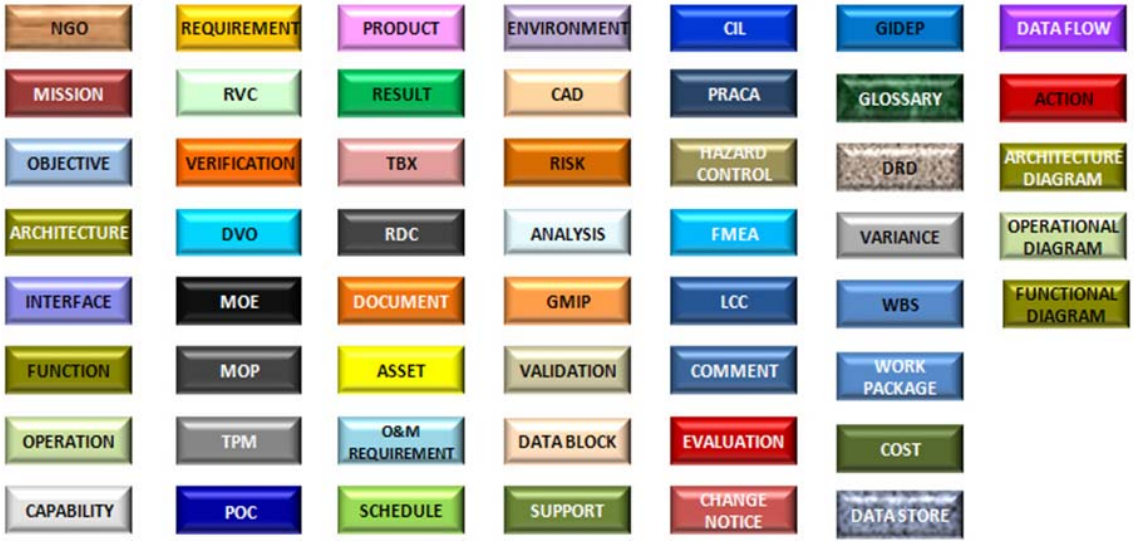

Figure 2 - Sample Data Object Diagram

Data Attributes include the metadata that describes and defines the data objects. The identification of the Data Attributes should be directly related to the purpose of the data object and NOT the integrated report/viewpoint. Many times Data Architectures and Data Object attribute schemas are developed based on the contents of a desired report. This practice causes data redundancy, multiple dependencies and implementations. When this occurs the data quality is compromised and must be reconciled and synced until the DA can be normalized. When the Application and methodology are developed the Data Architecture can be parsed and implemented as desired, crossing Data Object attribute boundaries, assuming duplication does not occur. Examples of Requirement Data Object attributes are Requirement Statement, Requirement Rationale, and Requirement Type. Each of these attributes, with respect to the Requirement Data Object will need to be defined in the DAF with the expectations of their use. For instance, is the attribute Mandatory and must be populated prior to full maturation? Or, is the attribute optional and at the discretion of the author? Or, perhaps the attribute is Conditional and must be provided logically based on another attribute or event?

Common Attributes, inherited by all Data Objects in the Data Architecture, should be defined and used for standardization of Life-cycle maturation, Configuration Control levels, data source identification, attribute formatting, etc. Examples of a common attributes that crosses multiple Data Objects are; Maturity Status, Security Markings (i.e. SBU, ITAR, EAR), author, last modified date, etc.

Data Associations are the proverbial glue that created the Architecture of the data framework. They describe the link/relationship/association, interdependencies, between Data Objects. These associations can be directly related to the Data Object of Interest, or be the conduit for transitive relationships, via a directly related Data Object. These associations can be uni- or bi-directional, to allow full traversing of the data architecture. Action semantics typically are defined for each relationship, ignoring the application architecture capabilities. Examples of semantics are Verifies, Specifies, Controls, Resolves, Causes, etc. The semantics between that describe the association should be comprised of no more than three words. There are several cases where two Data Objects can be directly related, but have multiple purposes. This is where a "role-based" type semantic describing the association differentiates the implied message from the relationship. An individual can be an owner, stakeholder or reviewer of a Requirement Data Objects, albeit not at the same time, but for different defined items. To keep the Data Architecture normalized the Organizational component as and the Requirement Data Objects have their own metadata, then the association provides the semantic(s) to define the relationship. These associations are very important when trying to "re-integrate" the data. The Data Objects and association semantics should tell a story as you trace from one data element to the next in simple sentence form. (See Figure 3 – Data Object Association Peacock Chart)

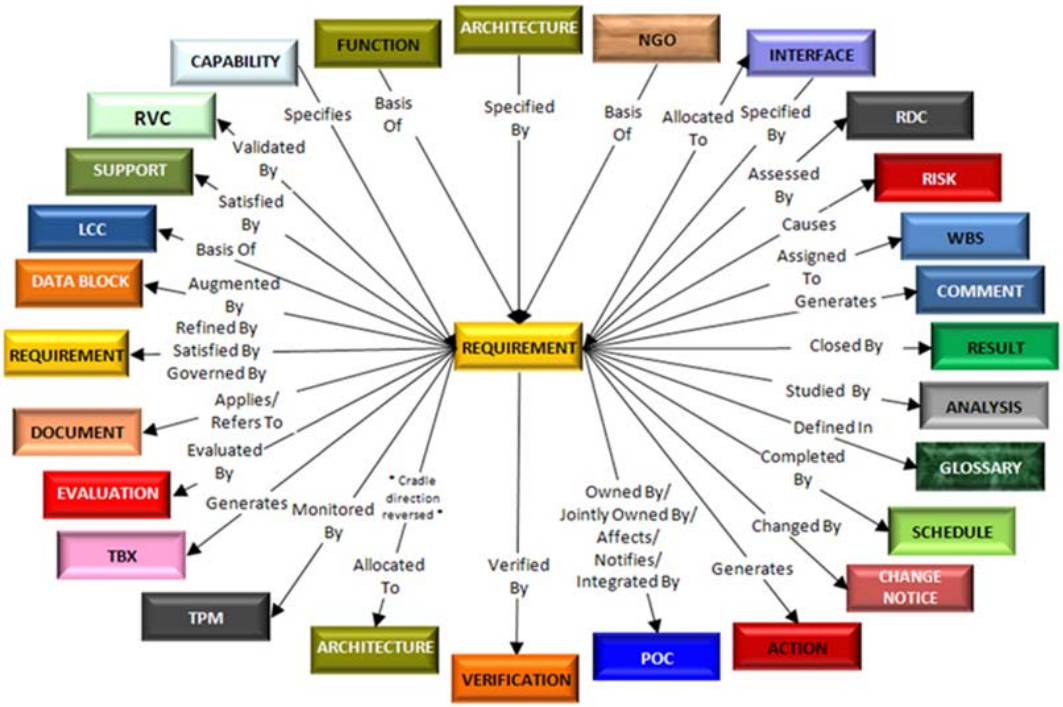

Figure 3 - Data Object Association Peacock Chart

Data Quality Audits can be defined to provide an automated way to ensure compliance with the defined Process-Based Data Architecture. These audits can be qualitative or quantitative in nature and can provide metrics and

information to ensure adherence to the processes and procedures and to measure the maturity level of the program/project as it moves through it's life-cycle phases. These data quality audits will be used when implementing your MBSE approach and help identify discrepancies as they occur per the defined measurements. Examples of audits include: Requirements with only one shall in the requirement statement, avoids compound requirements; Requirements with only one parent, helps identify duplicate requirements and potential traceability errors; Requirements that are more mature than their parent Requirement.

In a typical Document Based system the development of many of these data structures are stove-piped, with minimal consideration to related Data Objects. At best there are cross-reference matrices hidden away in appendices, which are rarely validated when a document is released.

3- Application Architecture Selection

Identify Authoritative Data Repository Application for Data Objects. Applications MUST be able to "expose" data to other Applications for Data Integration and Interrogation. These applications are typically server-based, not desktop instances. Applications are typically related to a specific Systems Engineering or Program Management function (i.e. Requirements Management, Risk Management, Schedule Management, Budget Management, etc.).

There are several broad spectrum applications that are on the market that can satisfy multiple functions, making data integration a less burdensome task. For smaller programs/projects this approach is preferred to limit the overhead of developing complex Information Technology Systems to integrate the disparate data.

4- Data Object to Application Assignment

The identified Data Objects, from the Data Architecture "Framework", necessary to perform the Project Management and Systems Engineering functions, per the Business Architecture, should be assigned to their authoritative repository (application). Typically authoritative data objects and attributes reside in only one application. The defined Data Objects are mapped to their respective application until all data objects have been assigned to at least one application. If data objects are assigned to multiple applications, then the authoritative source application must be identified. This practice will help to avoid stale, out of sync duplicative (reflective) data that will evidentially require the laborious task of normalizing and reconciliation.

5- Application Schema Development

Each application needs to incorporate the gerrymandered subset of the Data Architecture "Framework" based on the assignment of Data Objects to Authoritative Applications. The Program/Project can take liberties with the Data Object names and attribute labels; however the underlining definition must be maintained to ensure data integration. An XML schema, or other common format, for the Data Architecture Framework can be tailored and parsed, based on Program and application specifications and used to establish the traceability to the Data Architecture Framework and established processes.

6- Application Methodology Development

A Methodology, how you implement the process, should be developed and documented for data objects, internal associations and attributes for the assigned application. Each application will include it's own quarks and limitations related to establishing the methodology for capturing the necessary data. In theory, there should not be duplications of implementations, multiple ways of capturing the same data. This methodology needs to be governed to ensure the application's model artifacts are consistent, which can be accomplished in part by implementing the defined data audits.

7- Disparate Data Objects Identification

An analysis of the Application Architecture and their assigned Data Objects will result in the level of data disparity within the program/project. The Program/Project should document anywhere there is an association between two data objects that were assigned to different authoritative applications. Ideally a Program or Project would try to minimize the amount of disparate data objects to reduce the risk of reporting erroneous or stale information. For example, if Requirements are in a Requirements Management application and Risks are in a Risk Management Application, there would need to be some way to associate the Risks "caused by" (uni-directional semantic example) a Requirement. You want to avoid putting the Requirements in the Risk Application and the Risk in the Requirements application as an integration solution if at all possible.

8- Data Integration Framework Definition

The purpose of the Data Integration Framework (DIF) is to perform as the middle-ware, which joins the disparate data object's assigned authoritative applications to maintain the structure of the data model. The DIF is the back-end Application-to-Application interface. The front-end of the DIF provides a user interface to record the associations (or link repository) across data objects/applications and to generate user defined reports and analysis. The Data Integration Framework should comply with the previously defined Data Architecture Framework to develop a Common Model. This common model will facilitate the transformation of data from Application-to-Application and generating reports.

The Common Model Data Architecture Framework provides the target environment for the source(s) from the application(s). The source application schema maps to the

common model schema based on the pre-defined Data Object attributes. This practice allows the source applications to change over time, but maintains the integrity of the common data model. If an application changes the only thing that should need to change is the mapping between the target and source architectures.

There are many different data mapping software applications that can help maintain the relationships between the source(s) and the target.

9- Technology Architecture Development

The Technology Architecture includes the Information Technology (IT) infrastructure, which includes the Data Integration Framework (DIF). To be successful the Technology Architecture must have real-time or near real-time access to the exposed application data items.

10- Define the System of Interest Architecture

After the Business, Data, Application, and Technology Architectures have been defined (or sufficiently defined to begin a subset, just in time need of the SE Process) the Model-Based Project Management and Systems Engineering efforts can begin for the System-of-Interest Architecture. It is important for the System-of-interest to be modeled per the described data architecture and application methodologies. Inter-application data items should be registered in the Data Integration Framework to extend the data model from one application to the other application(s) within the Technology Architecture. A tool agnostic SE Process Overview example is included in Section 7.

6. TRANSITIONING FROM DBSE

Transitioning an organization from a Document-Based Systems Engineering (DBSE) methodology (Implementation of the SE process using distributed documents, often duplicating data) to a Model/Data-centric (Implementation using authoritative, distributed, cohesive, tools/applications, object-oriented, define once use many times) methodology does not alter the underlying, in situ, well understood processes. A thorough understanding of the Big Picture, without the complexity of tool specific nuisances, will provide a bridge to adapting to the change to MBSE. This is what the Data Architecture delivers.

7. PROCESS OVERVIEW EXAMPLE

This process describes examples of major steps to fulfill the mandatory compliance requirements per Systems Engineering [2] and Program/Project Management Processes and Requirements [3]. They include a tool agnostic Data Architecture Framework overview of the integrated processes and steps necessary to implement the Program/Project SE and PM in an MBSE environment. The Program/Project will be required to parse out Data Objects and common sub-process into authoritative application repositories (ex. 3SL Cradle, IBM DOORS, Vitech CORE, SysML variants, PDLM, etc.) as documented in the Systems Engineering Management Plan.

The Data Objects Attributes, and Associations, previously discussed, can be identified while reviewing this example process. The diagram in Figure 4 – Integrated Data Objects

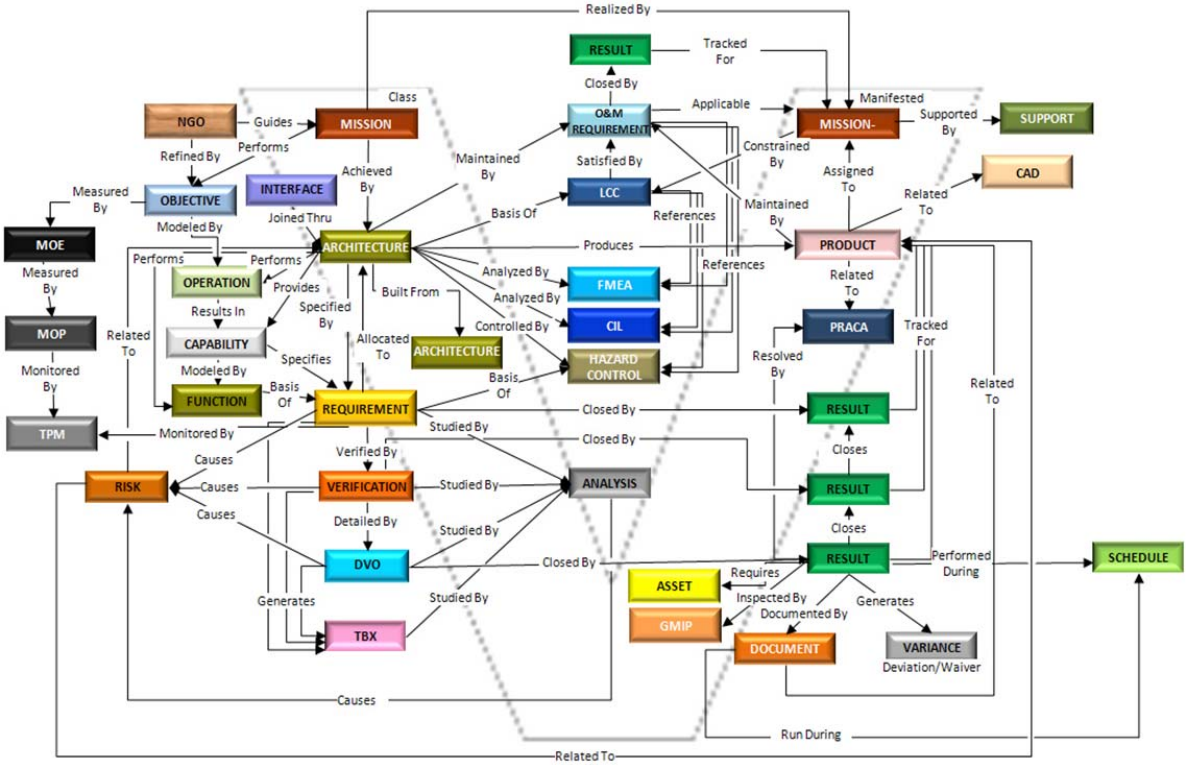

Figure 4 - Integrated Data Objects Partial Entity Relational Diagram (ERD)

Partial Entity Relational Diagram (ERD) represents the steps outlined in the process overview.

The Data Architecture Framework (DAF) and procedures provide the formalized Model-Based Systems Engineering (MBSE) application of modeling to support architecture development for System-of-Systems (SoS), System, and lower level architecture requirements, design, analysis, verification and validation activities. Life-cycle coverage implementation for the above begins in the conceptual design phase and continues throughout development of Design Reference Missions (DRMs) and later life cycle phases, including Mission/Operational Realization.

The Mission Based Configurable Architecture defined by this process includes the tangible, delivered Configurable End Products, which will be managed using Product Life-cycle Management (PLM) and Product Data Manager (PDM) processes. This approach addresses the top three levels of definition of the integrated Architecture, but could apply recursively with minor modifications, to any level. The process for “non-recursive” architecture layers are included:

Level 0 - Stakeholders

Level 1 - Directorate

Level 2 - Program or System-of-System (SoS)

Level 3 - System

Level 4 - Element (recursive)

Level 5 - SubSystem (recursive)

Level 6 - Component (recursive)

Level 7 - Part (recursive)

Note: Each level of the Architecture Hierarchy is the "System-of-Interest" for that level. The end Product "System" does not have to be delivered at the Architecture "System" Level. Within the Product Breakdown Structure (PBS) the "System" is the System, but may trace back to a different labeled layer of the Architecture.

The Directorate Architecture (Level 1) is initiated during the Formulation Phase of the lifecycle by developing the Formulation Authorization model. The program Formulation Authorization Model authorizes a program manager to initiate the planning of a new program and to perform the analysis of alternatives required to formulate a sound Program Plan that contains project elements, mission, goals and objectives, organization breakdown structures, requirements, schedules, risk assessments, and budgets. Based on these Needs, Goals and Objectives (NGOs) the Directorate Level determines the necessary Design Reference Mission(s) and associated Objectives to meet the associated NGOs. The Directorate level develops it’s plans to perform Program and Systems Engineering Management functions along with an associated Master Milestone Schedule.

If required, a Formulation Agreement (FA) is prepared by the project as a response to the Formulation Authorization to establish the technical and acquisition work that must be conducted during Formulation phase, which defines the schedule and funding requirements..

Based on Stakeholder NGOs, the Design Reference Mission (DRM), and Objectives the Directorate (Level 1) develops DRM Objective based Operational Concepts (Concepts of Operation) models. The DRM operational model is developed in stages and levels in a "divide and conquer" fashion to fully define the execution flow of the mission to capture the order of Operational Phases. The Operational Phases are decomposed into Operational Segments, and the Segments into Operational Activities.

This Operational Concept definition provides necessary details for Subject Matter Experts (SME) to derive a set of Architectural Capabilities necessary to support the realization of the DRM. (Note: Mission based Operational Models, however, may also be developed at lower "intangible" Architecture Levels (i.e. Program or System Level) if further operational decomposition is required.) A capability can be derived by an observed operational need for an entire mission phase, sub-phase, an activity, event, an operation or set of operations. These capabilities are allocated to the Program Architecture. This will eventually define the capabilities the Program’s end Products will be required to provide. These Mission related capabilities, along with their associated operations, are used to determine the next lower level (Program) Architecture required to fulfill the objectives and operational needs of the DRM. From the operational capabilities Directorate Architecture Level Requirements are developed and traced to the Operational Capabilities, as required. These capability based requirements are transitively traceable and should support the Operational Concepts, Mission Objectives, and NGOs. The Directorate Level specified Requirements are allocated to the derived Level 2 Program Architecture.

At the Directorate Level, and as required by the Program Level, Measures Of Effectiveness (MOEs), based on Mission Objectives, are used to qualitatively measure critical stakeholder's operational/mission expectations. Generally, each MOE has two or more quantitative Measures of Performances (MOPs) to ensure the associated MOE is met. From a defined set of MOPs a set of critical or Key Performance Parameters, Technical Performance Measurements (TPMs) are developed to track and measure progress and identify deficiencies that might jeopardize the implementation of related requirements.

The Program Architecture (Level 2) is initiated by the instantiation and allocation of the Directorate Architecture (Level 1) Requirements. This allocation is dependent on first defining the Mission, Objectives, and Program Architecture (Level 2), which represents the objectives, capabilities, and subsequent functionality for a deliverable set of Architecture Products to meet those constraints. Systems, Organizations, and/or Services external to the

Program, which interface with the Program Architecture Level, are identified from the allocated Directorate Architecture Requirements. From this set of allocated Directorate Architecture requirements a set of Program Architecture specified requirements are created and traced to their parent Directorate requirements. The remainder of the Program's specified Requirements are developed from allocated constraint requirements from the Directorate Level and any derived requirement based on the definition of the Architecture.. The Program model allocated and specified competencies are evaluated and Functional, Performance, Interface (External/Internal), Constraint and other areas of expertise requirements are created to specify the Architecture level. Operational events, resulting from transitions from one Operational Phase to the next), or Operations, which require interaction between entities, are identified and modeled as Architecture Mission Configurations.

Analysis is performed on all Program specific Requirements and related model meta data, to identify/derive the System Architecture required to perform the objectives of the Program. The Program's specified Requirements are then allocated to the defined System Architecture. This process continues until all Systems have been identified and all necessary Program applicable Requirements have been allocated to the System Architecture.

Interfaces are derived by analyzing the Program Requirements that have been allocated to more than one System and determining if there is a structural or interaction between the allocated Architectures. When an Interface has been identified between two or more Systems the interaction between the involved Systems will be evaluated and the Interface will be established and characterized. The Characterization of the Interface will define it's attributes, features, and constraints. The Characterized Interfaces will be decomposed with increased definition and details for the interface. These Characterized interfaces will be the basis for defining the set of System-to-System applicable Interface Requirements. Requirements to another System) and allocated down each architecture level to the point of implementation. Each level analyzes the composite requirement's associated requirements for implementation. If the analyzing architecture level is implementing one or more interface requirements, then a child requirement to the parent interface requirement is written and specified to that architecture level.

The Program Requirements allocated to the Systems will be evaluated and specific System Requirements will be created. Operational Analysis for Concept of Operations will be modeled for based on each System's allocated Operation. The System's allocated Operations are analyzed and applicable Operational/Capability based System Requirements are developed. Functional Analysis, modeled using Functional Flow Block Diagrams (FFBD), will be performed for each System by evaluating the System specific requirements to identify it's major functions. These major functions and any identified/derived functional Interfaces are modeled. Derived Functions are allocated to the System Architecture identified to perform the function(s). The System's allocated Functions are evaluated and a set of System Architecture applicable Functional Requirements are created. These function derived requirements are traced to associated Program (Level 2) Architecture Requirements, as parents, and also to the associated functions. In some cases (i.e. intangible architecture levels), Operational modeling and Capabilities will be performed at the System Level. Functional Analysis should be performed at a tangible Architecture level….

There are several Data Objects and associations you should have identified while reviewing this excerpt; Mission, Architecture, Function, Requirement, NGOs, Objectives, Capabilities, MOEs, MOPs, Interfaces, TPMs, etc.

Additionally, many associations should have been derived from the process; Operations provide Capabilities, Capabilities to Functions, Functions to Requirements, Requirements specifying Architecture, Requirements allocated to Architectures, Objectives measured by Measurements of Performance, etc.

Data Object attributes such as Requirement Types (Functional, Operational, Performance, etc.) are also included in the overview.

These are Data Architecture elements that can be extracted from the processes and used to document the tool agnostic common model.

8. SUMMARY

Once we have defined what MBSE represents we can translate it into many different “languages”. The same is true with developing a tool agnostic, process based Data Architecture. Once it is defined it can be represented into many different applications (languages) and translated in tool agnostic representation for mapping and transformation.

Data Object metadata

Attributes - A quality or characteristic inherent in or associated with a business process (ex. String, Fixed-String (16/60/256/etc.), Enumeration (Category), Single Pick (sp), Multiple Pick (mp), Boolean/File, Date (MM/DD/YYYY)

Association - Definition of relationships between Data Objects, Bi or Uni-Directional semantics, From/To, Parent/Child hierarchy

Audits/Metrics - Measurements of Process Compliance, Quality of Data, Measurement of Goals

The Data Architecture is Object-Oriented and once developed only requires minimal maintenance and upkeep. This process should also fulfill ISO 9001 Quality Management Systems (QMS) requirements. Say what you do, do what you say, prove it!

REFERENCES

[1] ISO/IEC/IEEE 42010:2011 - Systems and Software Engineering – Architecture Description http://www.iso-architecture.org/ieee-1471/

[2] NASA NPR 7123.1 Systems Engineering Processes and Requirements

[3] NASA NPR 7120.5 Program and Project Management Processes and Requirements

BIOGRAPHY

Robert Crain *is a Certified Systems Engineering Professional (CSEP) endorsed by INCOSE (International Council on Systems Engineering), a United States Navy veteran qualified in SSN and SSBN submarines and has over 25 years of Systems Engineering experience with 22 of those years working four major NASA contracts (SpaceLab, The International Space Station (ISS), Constellation, and currently Exploration). During his tenor he has performed various functions including Project Management, Command and Data Handling Engineering, Integration Engineering, Requirements Management, and Model Based Systems Engineering Development. He was nominated for the Rotary National Award for Space Achievement (RNASA) Stellar Award for MBSE methodology development. Mr. Crain was awarded a NASA Silver Snoopy Award for his efforts related to the development of the ISS Flight Emulator, which was used to test Space Station Elements on the Ground prior to Flight. Additionally, he has instructed courses in AC/DC and Digital Electronic Theory. He is currently a senior Software Engineer for MEI Technologies Corporation.*