Architecture Definition Guide

(DoDAF v2.0)

Vitech

Vitech Corporation
2270 Kraft Drive, Suite 1600
Blacksburg, Virginia 24060
540.951.3322 FAX: 540.951.8222
Customer Support: support@vitechcorp.com
www.vitechcorp.com

Publication Date: August 2010

Table of Contents

List of Figures

List of Tables

Preface

This Architecture Definition Guide (ADG) provides a structured approach for populating a CORE project with architectural definition information using the Department of Defense Architecture Framework (DoDAF) schema provided with CORE. For detailed information about the DoDAF, refer to the Department of Defense Architecture Framework Version 2.0, 28 May 2010 (Volume 1, Volume 2, and Volume 3). This guide is written as a supplement to the CORE System Definition Guide (SDG)[1].

An architecture contains both operational elements, system elements, and program management elements; therefore, enterprise and operational development must consider these three areas[2]. This ADG presents the activities required to capture and develop an operational architecture. Operational models are developed using model-based system engineering (MBSE) principles, which apply equally well to architecture development, and the engineering activities integrate the operational model and the systems model. These architectural developmental activities may be expressed in terms of system engineering domain activities without loss of specificity or generality. These system engineering domain activities consist of operations/requirements analysis, functional analysis, physical architecture synthesis, and design verification and validation. An overview of the MBSE process is portrayed below for reference. At all stages of architectural development, CORE can produce documentation for the purpose of presentation, review, and analysis of the architecture as well as integrate and compare other architectures. The DoDAF v2.0 viewpoints[3] become available as a consequence of applying MBSE to a specific operational architecture.

This guide describes each architectural development activity and the CORE DoDAF v2.0 schema classes used to capture the associated information along with a schema diagram and table, identifying the schema classes used when performing this activity. Following the engineering activity discussion, the associated attributes and relationships are also presented. In addressing each activity, attention is given to populating the database in a manner that facilitates the production of DoDAF v2.0 viewpoints using the standard scripts provided with CORE.

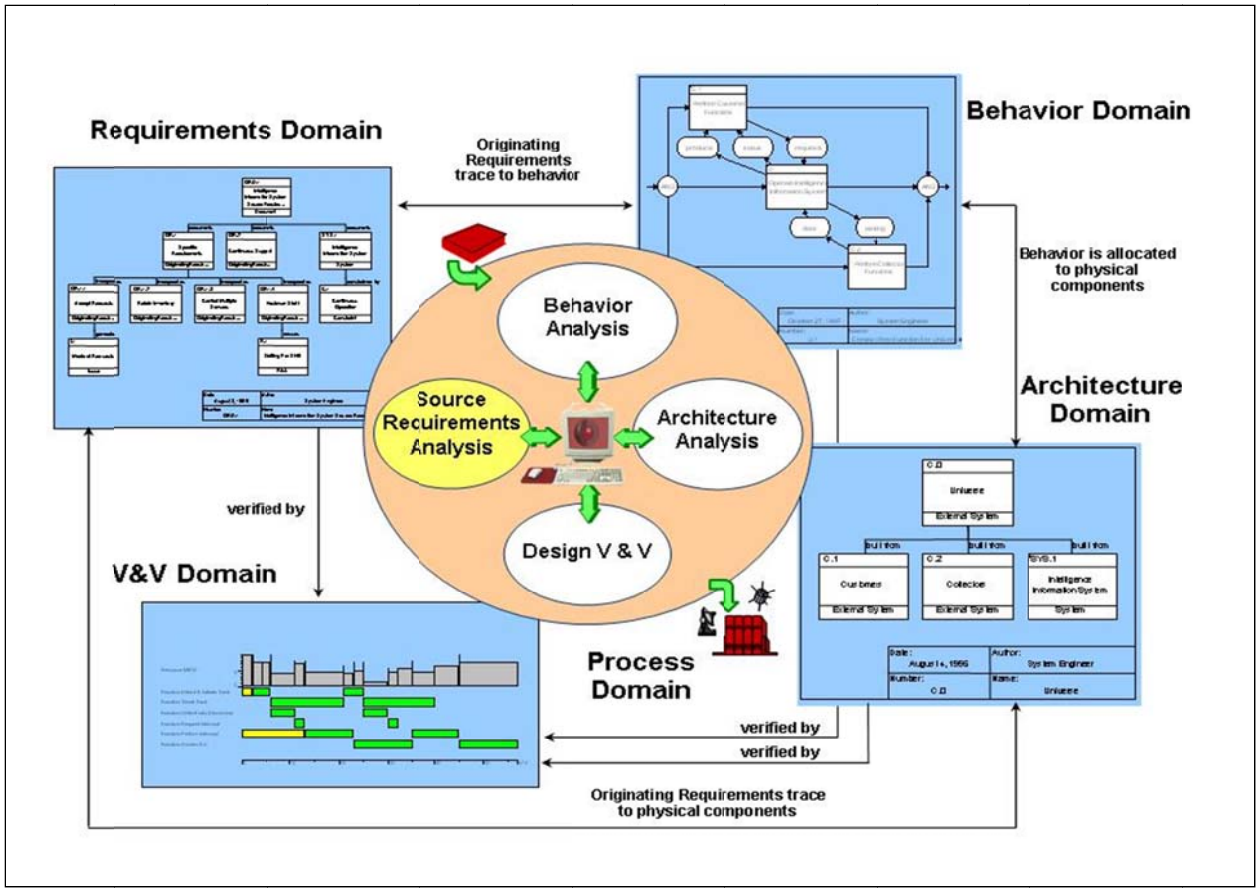
This guide augments the SDG and the Model-Based System Engineering with CORE® training course. The approach used here is generic and is not exhaustive of all cases and situations. This approach is written in the context of developing an operational definition before addressing the system definition. The programmatic aspects will also vary depending upon the state of the architecture, whether multiple architectures are being managed, etc. When working with "as-is" architectures, the activities may be reordered to best capture the existing as-is architecture.

---

[1] DoDAF v2.0 focuses on providing past, current, and future architectures for the executive/manager, while DoDAF v1.5 and its predecessors provide material from current and near-term IT architectures primarily for the program office. CORE's schema accommodates both perspectives.

[2] Enterprise architectures follow these same principles, however, enterprise architectures are not specifically address in this architecture definition guide.

[3] DoDAF v1.0/v1.5 views are also available for use with legacy architectural models and current models being developed under these previous versions of the DoDAF. Conversion of these models to DoDAF 2.0 models is possible.



MBSE Activities

The following additional resources are available for use with this guide:

- For descriptions of different behavior diagram notation, and the mechanics of entering data into CORE, the reader is referred to CORE's on-line help.
- For the definition of schema terms, the reader is referred to the CORE DoDAF v2.0 schema, which contains descriptions for each schema entity, and to the Schema Definition Report script that outputs complete documentation of the schema definition for a selected facility including descriptions for each schema entity.
- For details on generating DoDAF v2.0 viewpoints, the reader is referred to the script help file provided for each DoDAF v2.0 script. The user may access this documentation by selecting the Run Script icon on the CORE Explorer toolbar, selecting the DoDAF v20 folder, selecting any one of the DoDAF v2.0 scripts such as the (AV-1) Overview and Summary Information script, and pressing the Script Help File button. Note that CORE continues to support the DoDAF views developed to support DoDAF v1.0/v1.5. These report scripts reside in the DoDAF scripts folder.

Architecture Concepts

As portrayed in Figure 1 CORE's DoDAF v2.0 Schema – Part 1, CORE divides an architecture into an Operational Architecture Domain and a System Architecture Domain. The Operational Architecture Domain is used to capture originating concepts, capabilities, and the supporting operational analysis to expose the requirements leading to, and implemented in, the System Architecture Domain.

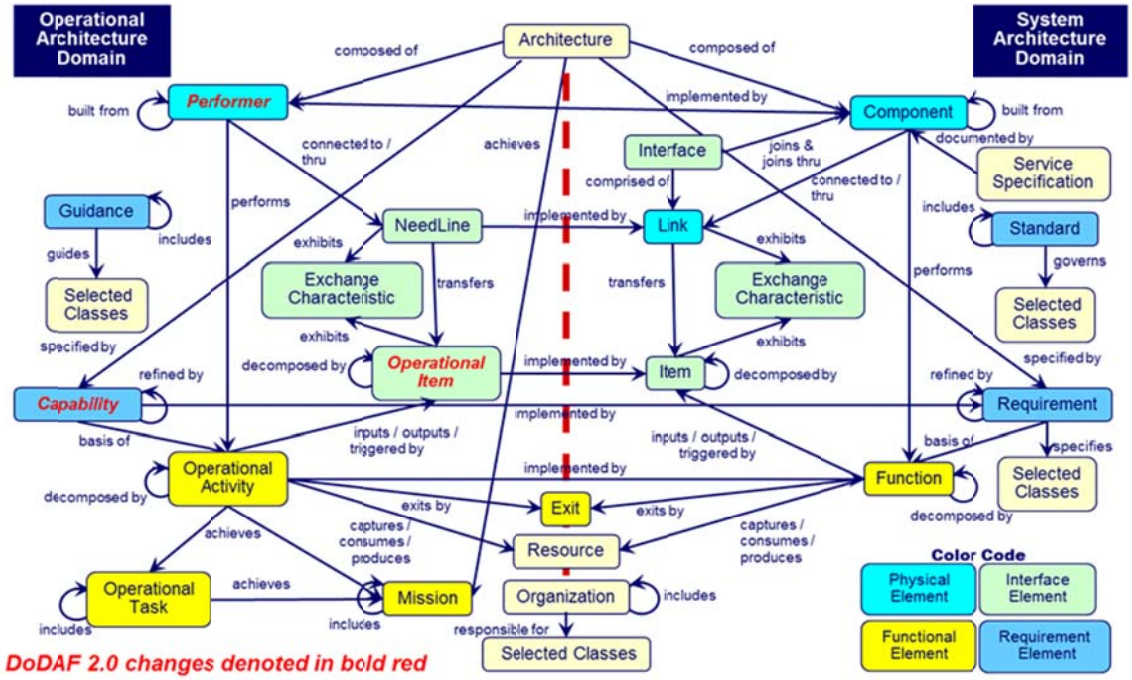


Figure 1 CORE's DoDAF v2.0 Schema – Part 1

As portrayed in Figure 1 CORE's DoDAF v2.0 Schema – Part 2, CORE integrates the Program Management Domain with both an Operational Architecture Domain and a System Architecture Domain. The Program Management Domain addresses the programmatic aspects of the architecture/system to assist in managing the current effort as well as finding commonality, duplicative, and missing capabilities among architectures. These aspects help an executive/ manager address duplication, misappropriation of scarce resources and the timeliness of the delivered capabilities to the enterprise.

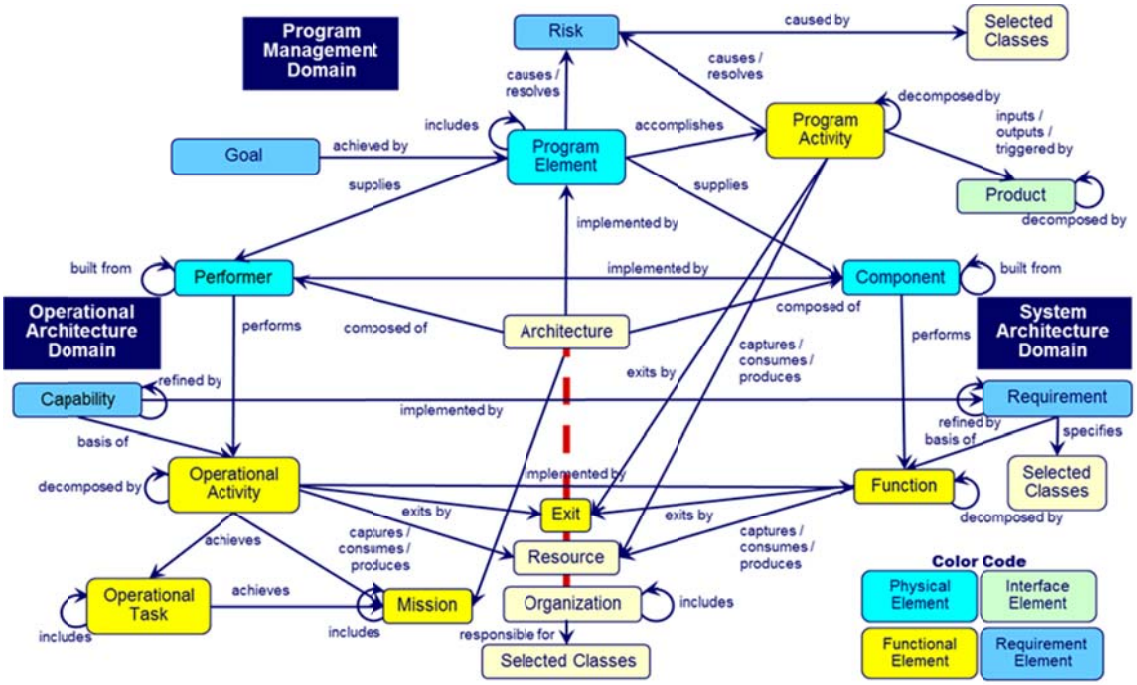


Figure 2 CORE's DoDAF v2.0 Schema – Part 2

This Architecture Definition Guide (DoDAF v2.0) provides guidance into structuring the elements, attributes and relationships that implement the Operational Architecture Domain and Program Management Domain for a project. Similarly, the System Definition Guide provides guidance into structuring the elements, attributes and relationships that implement the System Architecture Domain.

Operational and System Architecture Domain Relationships

The Operational Architecture Domain provides the necessary classes, attributes, and relationships to capture the foundational concepts, guidance, and the subsequent operational analysis to support define the interrelationships among architectures and systems along with documenting the source requirements for a system [or systems] of interest. The architecture element which spans the two domains is specified in paragraph *1.1 Define Architecture* and is composed of **Performer** (of type: Operational Architecture) and **Component** (of type: Family of Systems, Systems Architecture, or System of Systems) element(s).

Within the Operational Architecture Domain, the **Performer** (type: Operational Architecture) is part of the operational context which also includes the **Performer** element(s) that represent the external aspects of the operational domain. See paragraph *1.4 Define Operational Boundary* for details on defining the operational boundary.

Similarly, the System Architecture Domain includes the **Component** element (of type: Family of Systems, Systems Architecture, or System of Systems) which represents the system(s) of interest. This element forms part of the system context, which includes the **Component** element(s) representing the external aspects of the system domain. See *CORE System Definition Guide, paragraph 1.3 Define System Boundary* for details on defining the system boundary.

1 Operational Concept Capture

This section is written assuming that the customer or end-user has provided a Concept of Operations (CONOPS) or an operational capabilities or operational requirements document. If that is not the case, it is then assumed that the system/architectural engineering team will start with the task of collecting all stakeholder needs and transforming them into the required operational information. The end result of this effort will be a collection of requirements that are treated as originating operational requirements and/or architectural guidance information (See Section 1.2).

1.1 Define Architecture

Identify the architecture. Architectures exist for the purpose of achieving a well-defined system or more broadly for the enterprise, systems of systems (as defined in both the operational and system domains) for a specific time frame or time frames. The **Architecture** class is used to identify an architecture and its time frame. Each architecture is composed of an operational architecture and a systems architecture. Performers (operational nodes in DoDAF v1.5) in the operational architecture are represented in CORE using the **Performer** class. Physical entities, including collections of systems, interfacing systems, and entities within the systems architecture, are represented in CORE using the **Component** class. A **Performer's** or **Component's** Type attribute designates what the element represents (in this case an operational architecture for a **Performer** and systems architecture, system of systems, or family of systems for a **Component**). The Type attribute may indicate the role of the element or its relative position within the performer hierarchy.

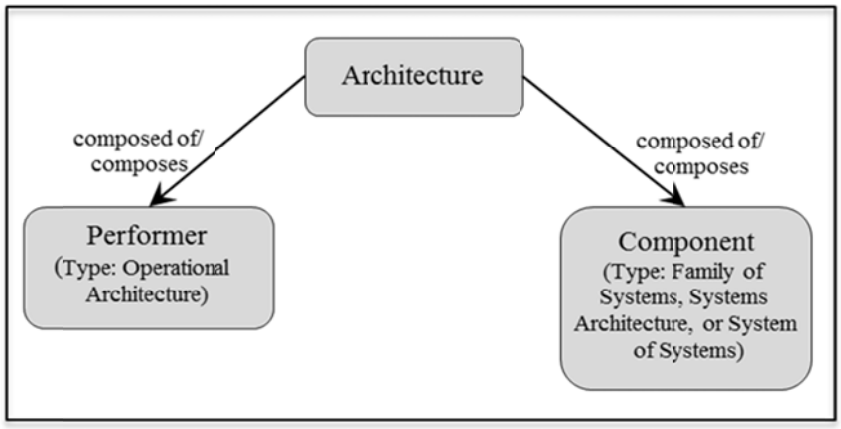


Figure 3 Architecture Definition[4]

[4] The relationships presented in this figure and the following are not exhaustive but to show the primary relationships for the topical area.

Table 1 Architecture Definition

Element Class	Attributes	Relationships	Target Classes
Architecture	Description Number Purpose Scope Time Frame[5]	*composed of / composes*	**Component** **Performer**
Component	See SDG Type: Family of Systems, Systems Architecture, or System of Systems	*composes / composed of*	**Architecture**
Performer	Abbreviation Cost Description Doc. PUID Latitude Location[6] Longitude Number Purpose Type: Operational Architecture	*composes / composed of*	**Architecture**

1.2 Capture Source Material

Capturing source material involves the creation of the following entries in the database depending on the information provided or needed:

- **Capability** element for each source capability statement[7]
- **Document** element for each source document
- **Guidance** element for each source statement that is not a mission, operational task, or requirement statement
- **Mission** element for each pertinent mission area or description

[5] It is recommended that the **Architecture** for each distinct time frame be captured in separate CORE Projects.

[6] The *Location* attribute provides a means of specifying physical and logical locations (addresses) in conjunction with physical latitude and longitude or independent of latitude and longitude.

[7] A *Capability Requirement* is distinguished from a *Capability* and is placed in the **Requirement** class with the type attribute set to *Capability*

- **OperationalTask** element for each operational task from a source such as the Universal Joint Task List (UJTL) or the Mission Essential Task List (METL)[8]
- **Requirement** element for each source requirement[9]
- **ExternalFile** element for each source guidance, requirement, mission, or operational task-related table or graphic
- **DefinedTerm** element for each pertinent acronym or special term in the source documents

As part of the process of capturing source material, the following should be done:

- Place any tables and graphics in separate files and reference them in the project database using **ExternalFile** elements where each *augments* the subject element. The formal documentation scripts, as well as the Architecture Description Document (ADD) and System Description Document (SDD) scripts, will automatically include these external tables and graphics in the output immediately following the element Description and make entries in the List of Figures and List of Tables, as appropriate. In order to properly number and label the tables and graphics for inclusion in the output, only a single graphic or table should appear in each file.
- Acronyms and/or special terms appearing in the source document should be captured in the database as **DefinedTerms**. For an acronym or abbreviation, the acronym is entered into the Acronym attribute and what it stands for is entered as the name of the element. For a special term, the term is the name of the element and its definition is entered into the Description attribute. By filling in both the Acronym and Description attributes, appropriate entries will appear in both the acronym and glossary sections of the ADD.

Extracting elements from source documents. The entry of source elements into a CORE database may be accomplished by using one or more of the following:

- Element Extractor window
- Document/Shall Parser script if extracting requirements
- Advanced CSV File Parser script if the elements are being transferred as a CSV file from another application such as IBM® Rational® DOORS®, Microsoft Excel, or Microsoft Access
- Copy and Paste or Paste Unformatted commands.

[8] The **OperationalTask** class is only used in those instances where traceability from a source such as the UJTL or METL is required. These tasks are specified, not derived.

[9] Examples are architecture and operational constraints and task performance characterization.

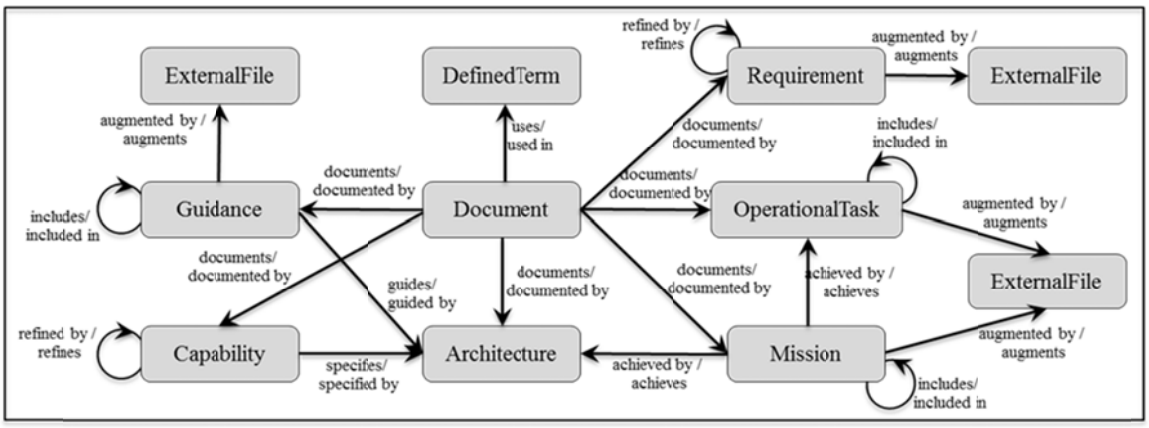


Figure 4 Source Material

Table 2 Source Material

Element Class	Attributes	Relationships	Target Classes
Architecture	See 1.1	*achieves / achieved by*	**Mission**
		augmented by / augments	**ExternalFile**
		composed of / composes	**Component** **Performer**
		documented by / documents	**Document**
		guided by / guides	**Guidance**
		implemented by / implements	**ProgramElement**
		specified by / specifies	**Capability** **Requirement**
Capability	Benefit Description Doc. PUID Key Performance Parameter Origin Paragraph Number Paragraph Title Rationale	*augmented by / augments*	**ExternalFile**
		basis of / based on	**OperationalActivity**
		documented by / documents	**Document**
		guided by / guides	**Guidance**
		implemented by / implements	**Requirement**
		provided by/ provides	**ProgramElement**
		refined by / refines	**Capability**
		refines / refined by	**Capability**
		specified b y /	**Requirement**

Table 2 Source Material

Element Class	Attributes	Relationships	Target Classes
		specifies	**Architecture** **Interface** **Needline** **OperationalItem** **Performer** **State/Mode**
		supplied by / supplies	**ProgramElement**
DefinedTerm	Acronym Description	*used in / uses*	**Document**
Document	CDRL Number Description Document Date Document Number Govt. Category Non-Govt. Category External File Path Number Type	*documents / documented by*[10]	**Architecture** **Guidance** **Mission** **OperationalTask** **Requirement**
		uses / used in	**DefinedTerm**
ExternalFile	Description External File Path Number Page Orientation Title Type	*augments / augmented by*[11]	**Guidance** **Mission** **OperationalTask** **Requirement**
Guidance	Description Number Paragraph Number[12] Paragraph Title[6] Type	*augmented by / augments*[5]	**ExternalFile**
		documented by / documents	**Document**
		guides / guided by	**Architecture**
		included in / includes	**Guidance**
		includes / included in	**Guidance**
Mission	Description	*achieved by / achieves*	**Architecture** **OperationalTask**

[10] Only the top-level **Guidance**, **Mission**, **OperationalTask**, and **Requirement** elements need to be *documented by* the source **Document.**

[11] The Position attribute of this relationship should be set to control the order in which multiple external files are appended to the element's Description attribute when it is output in the ADD.

[12] Used to record the source document paragraph number and title.

Table 2 Source Material

Element Class	Attributes	Relationships	Target Classes
	Number	*augmented by / augments*[5]	**ExternalFile**
		documented by / documents	**Document**
		guides / guided by	**Architecture**
		included in / includes	**Mission**
		includes / included in	**Mission**
OperationalTask	Description Number	*achieves / achieved by*	**Mission**
		augmented by / augments[5]	**ExternalFile**
		documented by / documents	**Document**
		included in / includes	**OperationalTask**
		includes / included in	**OperationalTask**
Requirement	Description Doc. PUID Key Performance Parameter Incentive Performance Parameter[13] Number Origin: Operational Paragraph Number[6] Paragraph Title[6] Rationale Units Value Weight Factor	*augmented by / augments*[5]	**ExternalFile**
		documented by / documents	**Document**
		refined by / refines	**Requirement**
		refines / refined by	**Requirement**

Warning: *The default font for text attributes, such as Description, is Times New Roman 10. Within a text attribute, the user has control over color, fonts, styling, sizing, and special effects such as underline, superscript, and strikethrough. The documentation scripts do not override any user modified fonts or special effects; however, they can override color, styling, and font size if the font is Times New Roman (they only control the styling of text in Times New Roman).*

[13] This parameter identifies the performance requirement or other requirement incentivized on a particular contract.

Consequently, in order to produce professional looking documents, care should be taken when capturing external source material. Specifically, when using the Element Extractor window, either turn off the Maintain Formatting option or pre-process the document to convert all text to Times New Roman (i.e., open the document in a word processor, select all contents of the document, and select Times New Roman as the font). Similarly, when using cut & paste, either pre-process the document to set the font to Times New Roman or use Paste Unformatted rather than the Paste command. Since they should not be modified on output, formulas should be captured in another font, such as Arial. Also, note that text attributes do not support embedded tables and graphics. Therefore, tables and graphics should be captured as ***ExternalFile*** *elements.*

1.3 Identify Organizations

Based on the source documents, identify the organizations that are key players in the architecture using elements in the **Organization** class. Capture the command structure as well as the coordination relationships among these organizations.

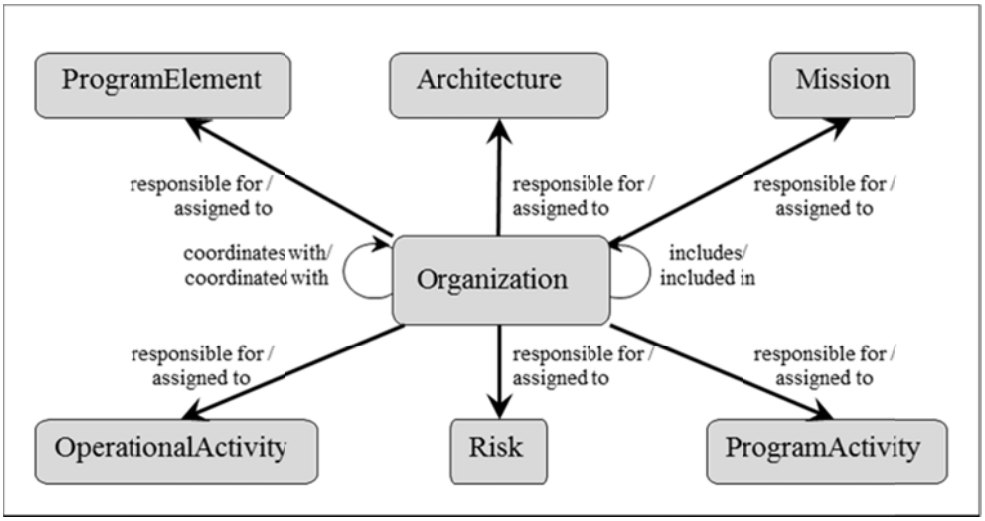


Figure 5 Organizations

Table 3 Organizations

Element Class	Attributes	Relationships	Target Classes
Organization	Abbreviation Description Latitude Location Longitude Number	*coordinated with/ coordinates with*	**Organization**
		coordinates with/ coordinated with	**Organization**
		included in/ includes	**Organization**
		includes/ included in	**Organization**

Table 3 Organizations

Element Class	Attributes	Relationships	Target Classes
	Role	*responsible for / assigned to*	**Architecture** **OperationalActivity** **Mission** **ProgramActivity** **ProgramElement** **Risk**

1.4 Define Operational Boundary

Based on an examination of the source, identify the operational boundary and context. To define the boundary, identify each operational external with which the architecture must interface. An operational external is represented as a **Performer** and may identify the operational environment. Create a **Performer** element representing the context and decompose it into the operational architecture and its externals using the *built from* relationship. Set the Type attribute for each **Performer**.

To complete the operational boundary definition, identify all the information exchanges between the architecture's performers and each external by creating elements of the **Needline** class. Defining a **Needline** element establishes that the architecture interacts with an external. Typically, there will be only one **Needline** between the architecture's performers and each external.

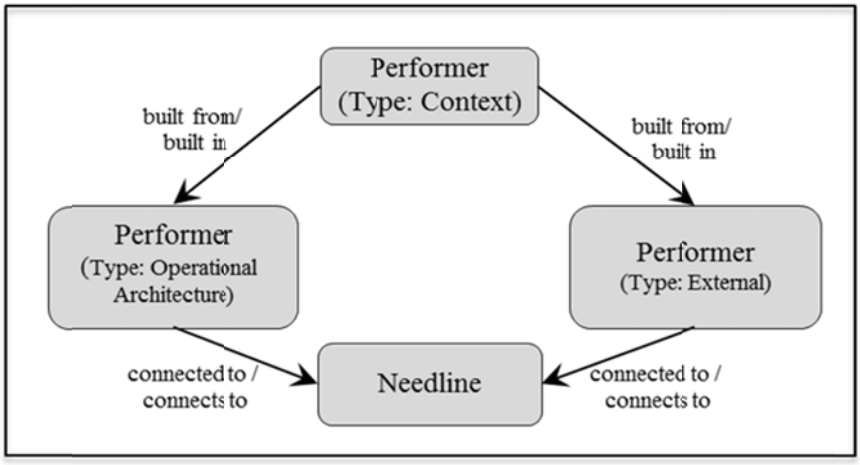


Figure 6 Operational Boundary

Table 4 Operational Boundary

Element Class	Attributes	Relationships	Target Classes
Performer (Type: Context)	Description Number Type: Context	*built from / built in*	**Performer** (Type: Operational Architecture and External)
Performer (Type: External)	Abbreviation Description Doc. PUID Number Purpose Type: External	*built in / built from*	**Performer** (Type: Context)
		connected to / connects to	**Needline**
Performer (Type: Operational Architecture)	See Section 1.1	*built in / built from*	**Performer** (Type: Context)
		connected to / connects to	**Needline**
Needline	Description Doc. PUID Number	*connects to / connected to*	**Component** (Type: External and Operational Architecture)

Suggestion: *Create a folder for the context and externals in order to separate them from the evolving performer hierarchy. Typically, the context and externals are given a different numbering scheme than the elements in the performer hierarchy in order to differentiate them in CORE views such as the Physical Block Diagram and Hierarchy diagrams.*

2 Operational Activity Analysis

Given the need to satisfy the operational mission(s) within the context of the CONOPS and/or the operational requirements document, the system engineering/architecture team must derive the necessary operational behavior for the operational architecture to accomplish the mission or missions. This is essentially a discovery process, working with operational activities to derive, define, or capture key capabilities. Finalized capabilities are integrated to become the integrated behavioral model for the architecture.

2.1 Operational Activity Model

Capabilities[14] form the foundation of an operational architecture. A capability is defined as:

> The ability to achieve a Desired Effect under specified [performance] standards and conditions through combinations of ways and means [activities and resources] to perform a set of activities.

Capabilities[15], in general, are the starting point for defining operational scenarios. These scenarios consist of a sequence of operational activities needed to respond to an external stimulus or to provide an external stimulus. **Capabilities** are the *basis of* **OperationalActivities** and are executable behavior entities. Each activity is *performed by* an element in the **Performer** class and the relationship attribute Behavior Type is set to "Capability". The integrated operational behavior is developed from integrating two or more capabilities into a single behavior model that fully represents the behavior required by a **Performer**. The relationship Behavior Type attribute for integrated behavior is set to "Integrated (Root)". Traceability between capabilities and the integrated operational behavior model is established through the *basis of* relationship. Logical groupings (taxonomy) of **capabilities** may be established through the *categorized by* relationship with elements within the class **Category**.

The context-level **OperationalActivity** is *performed by* the context-level **Performer** (of Type Context) with the relationship attribute Behavior Type is set to "Integrated (Root)".

OperationalActivity Inputs and Outputs. Each **OperationalActivity** within a capability or integrated behavior will have input and output **OperationalItem** elements identified. These **OperationalItem** elements are associated with **OperationalActivities** using the relationships: *input to/inputs*, *output from/outputs*, and *triggers/triggered by*. As with **OperationalActivities**, **OperationalItems** should be aggregated to simplify presentation.

OperationalActivity Assignment. In conjunction with Operational Architecture Synthesis (See Section 3.1), for each layer of **Performers**, **OperationalActivities** in the integrated behavior are decomposed until they can be uniquely assigned to the next level of **Performer** using the *performed by* relationship. This not only establishes the organization or role that performs the activity, it allows the system engineering/architecture team to assess the impact of **Performer**

[14] The usage of the term capability is as described in the DoD Architecture Framework, Version 2.0, 28 May 2009. In DoD oriented models, capabilities refer to operationally oriented scenarios and threads refer to system-oriented scenarios.

[15] There may be one or more *capability requirements* establishing the programmatic need and timeframe when the capability is needed. Capability Requirements are captured in the **Requirement** class of Type: Capability.

losses or failures on both **Mission** and **OperationalActivities**, thereby, making it easier for the system engineering/architecture team to design countermeasures to mitigate operational impacts of **Performer** loss or failure.

OperationalActivity Traceability. **OperationalActivity** traceability from an appropriate **Mission** element (or **OperationalTask** if required) is established using the *achieves* relationship. Establishing this relationship enables one to easily assess what capabilities and behavior are impacted by a **Mission** change, as well as answering the converse question of what **Missions** are impacted by a capability change or failure.

OperationalActivity traceability from an appropriate **Requirement** occurs in two senses. These relationships are the *specified by* and the *based on* relationships. The *specified by* relationship identifies constraint or performance requirements that the **OperationalActivity** must satisfy. The *based on* relationship is used for all other requirements that apply to the **OperationalActivity**.

OperationalActivity traceability from appropriate **Guidance** is established using the *guided by* relationship. This establishes traceability from any policy or doctrine that either constrains or requires an **OperationalActivity**.

Note: *When doing behavior modeling, a root* **OperationalActivity** *can be established for any* **Performer** *and the behavior diagram built using the assigned* **OperationalActivities** *to define the full behavior of the* **Performer** *from the* **Performer's** *perspective rather than from the operational architecture's perspective. These lower-level root* **OperationalActivities** *do not appear in the operational activity hierarchy, but act as tap points into the hierarchy.*

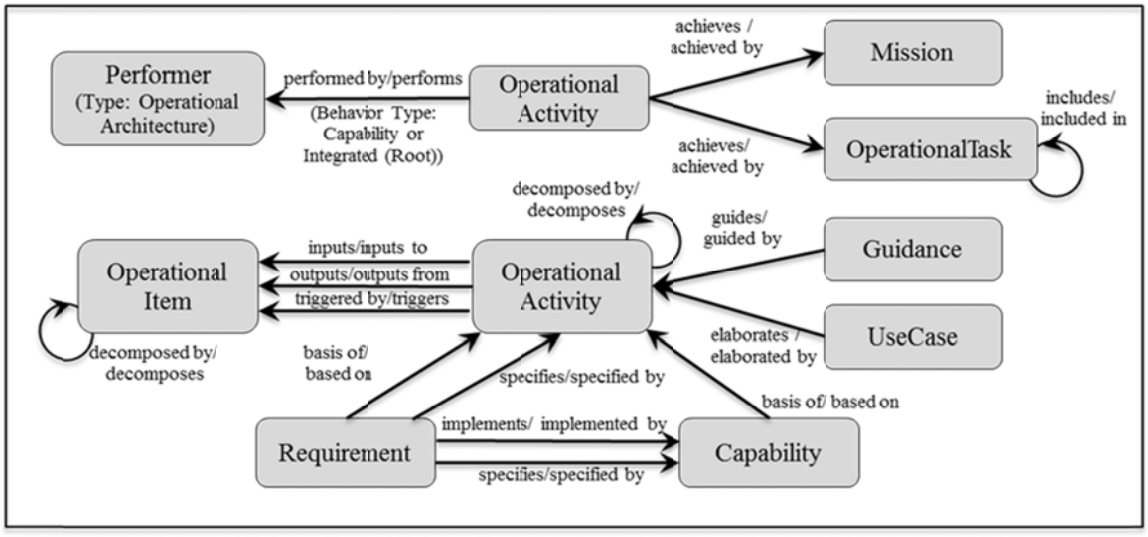


Figure 7 Operational Activity Model

Table 5 Operational Activity Model

Element Class	Attributes	Relationships	Target Classes
Capability	See Section 1.2	*based on / basis of*	**OperationalActivity**
Guidance	See Section 1.2	*guides / guided by*	**OperationalActivity**

Table 5 Operational Activity Model

Element Class	Attributes	Relationships	Target Classes
Mission	See Section 1.2	*achieved by / achieves*	**OperationalActivity**
Performer (Type: Operational Architecture)	See Section 1.1	*performs / performed by* (Behavior Type: Capability or Integrated (Root))[7]	**OperationalActivity**
OperationalActivity	Description Doc. PUID Duration Number	*achieves / achieved by*	**Mission** **OperationalTask**
		based on / basis of	**Requirement**
		basis of / based on	**OperationalActivity**
		based on / basis of	**OperationalActivity**
		based on / basis of	**Capability**
		decomposed by / decomposes	**OperationalActivity**
		decomposes / decomposed by	**OperationalActivity**
		elaborates /elaborated by	**UseCase**
		guided by / guides	**Guidance**
		inputs / input to	**OperationalItem**
		outputs / output from	**OperationalItem**
		performed by / performs (Behavior Type: Capability or Integrated (Root))[16]	**Performer**
		results in/ result of	**Capability**
		results in /result of	**Requirement**
		specified by / specifies	**Requirement**
		triggered by / triggers	**OperationalItem**

[16] An **Performer** could have multiple **OperationalActivities** of Behavior Type "Capability" but should have only one **OperationalActivity** of Behavior Type "Integrated (Root)".

Table 5 Operational Activity Model

Element Class	Attributes	Relationships	Target Classes
OperationalItem	Accuracy Description Doc. PUID Number Timeliness	*decomposed by / decomposes*	**OperationalItem**
		decomposes / decomposed by	**OperationalItem**
		guided by / guides	**Guidance**
		input to / inputs	**OperationalActivity**
		output from / outputs	**OperationalActivity**
		specified by / specifies	**Requirement**
		triggers / triggered by	**OperationalActivity**
OperationalTask	See Section 1.2	*achieved by / achieves*	**OperationalActivity**
Requirement	See Sections 1.2	*basis of / based on*	**OperationalActivity**
		specifies /specified by	**OperationalActivity** **OperationalItem**
UseCase	Alternate Flow Description Number Preconditions Primary Flow Postconditions	*describes / described by*	**Performer**
		elaborated by / elaborates	**OperationalActivity**
		extended by / extends	**UseCase**
		extends / extended by	**UseCase**
		generalization of / kind of	**UseCase**
		kind of / generalization of	**UseCase**
		included in / includes	**UseCase**
		includes / included in	**UseCase**
		involves / participates in	**Performer**
		specified by / specifies	**Requirement**

3 Operational Architecture Synthesis

3.1 Assign OperationalActivities to Next Level of Performers

In conjunction with the analysis of the CONOPS document, **OperationalActivity** as well as **Performer** decomposition occurs as part of the process to refine the operational architecture. This hierarchical decomposition process results in more specificity regarding subordinate **Performers** and the behavior that is required of them.

As the **Performer** hierarchy evolves, **Performers** uniquely *perform* more refined **OperationalActivities**. This is accomplished in layers. When a decomposed root or capability **OperationalActivity** is *performed by* a **Performer**, all lower-level **OperationalActivities** in its decomposition are part of the behavior of the **Performer**. The **Performer** may be correspondingly decomposed, in which case even lower-level **OperationalActivities** are performed by the lower-level **Performers**. These lower-level assignments are termed Atomic. Since **OperationalActivities** can be aggregated to enhance understanding, there is not necessarily a one-to-one correspondence between levels in the **OperationalActivity** hierarchy and levels in the **Performer** hierarchy.

Performers are mapped to **Organizations** using the *assigned to* relationship[17]. With all the previous relationships established as described in Section 2.1 for each layer of **Performer** decomposition, then it is possible, through tracing the appropriate relationships, to identify what capabilities and integrated behavior the **Organization** is responsible for as well as any subordinate **Missions**, if they were defined.

Note: *As stated in Section 2.1, when doing behavior modeling, a root* **OperationalActivity** *can be established for any* **Performer** *and the behavior diagram built using the atomic* **OperationalActivities** *to define the full behavior of the* **Performer** *from the* **Performer's** *perspective rather than from the architecture's perspective.*

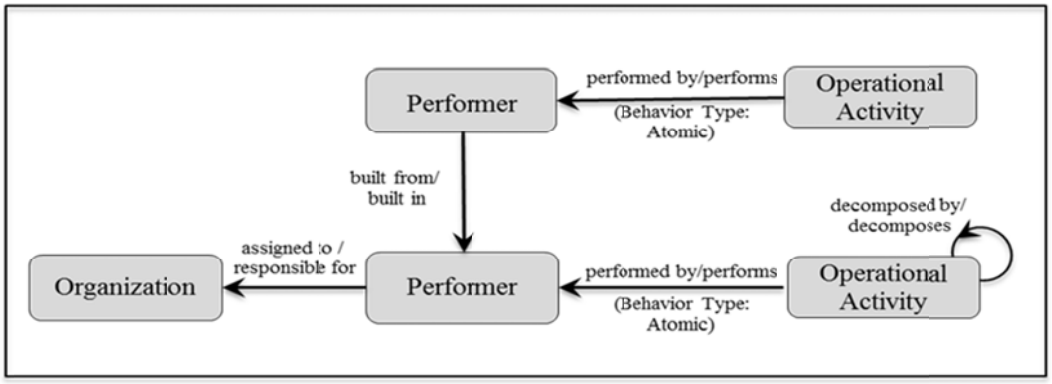


Figure 8 Performer Hierarchy and OperationalActivity Assignment

[17] Organizations, organizational units, roles, etc. are represented as **Organizations** elements with a parent-child relationship reflecting command structure. They are also represented as **Performers** in which case hierarchically related units are often peers because of the **OperationalActivities** that they perform and the communication need between them.

Table 6 Performer Hierarchy and OperationalActivity Assignment

Element Class	Attributes	Relationships	Target Classes
Performer	Abbreviation Description Doc. PUID Latitude Location Longitude Purpose Number Type	*assigned to / responsible for*	**Organization**
		built from / built in	**Performer**
		built in / built from	**Performer**
		performs / performed by	**OperationalActivity**
OperationalActivity	See Section 2.1	*performed by / performs*	**Performer**
Organization	See Section 1.3	*responsible for / assigned to*	**Performer**

3.2 Refine External Needline Definitions

An external **Needline** element identifies the fact that the operational architecture communicates in some manner with an external **Performer** (See Section 1.4)[18]. As the **Performer** hierarchy evolves, the terminus point for **Needline** is appropriately changed to lower-level **Performers** when the **Performers** that provide the **OperationalItem,** transferred by the **Needlines,** are determined by **OperationalActivity** assignment. When the target of a *connects to* relationship is changed from a **Performer** to one of its subordinates, CORE automatically establishes the *connected thru* relationship between the **Needline** and the parent of the subordinate **Performer**. This allows **Needlines** to retain their identity even though their end points may change as the **Performer** hierarchy grows in depth.

Needlines may be *specified by* performance and constraint **Requirements**. Only the lowest layer of **OperationalItem** should be *transferred by* a **Needline**.

[18] If the external **Performer** is a threat source, then the communication element offered by the threat source is some observable that an **OperationalActivity** within the **Architecture** can recognize. Including externals such as a threat source allows the engineering team to better analyze and specify the architecture.

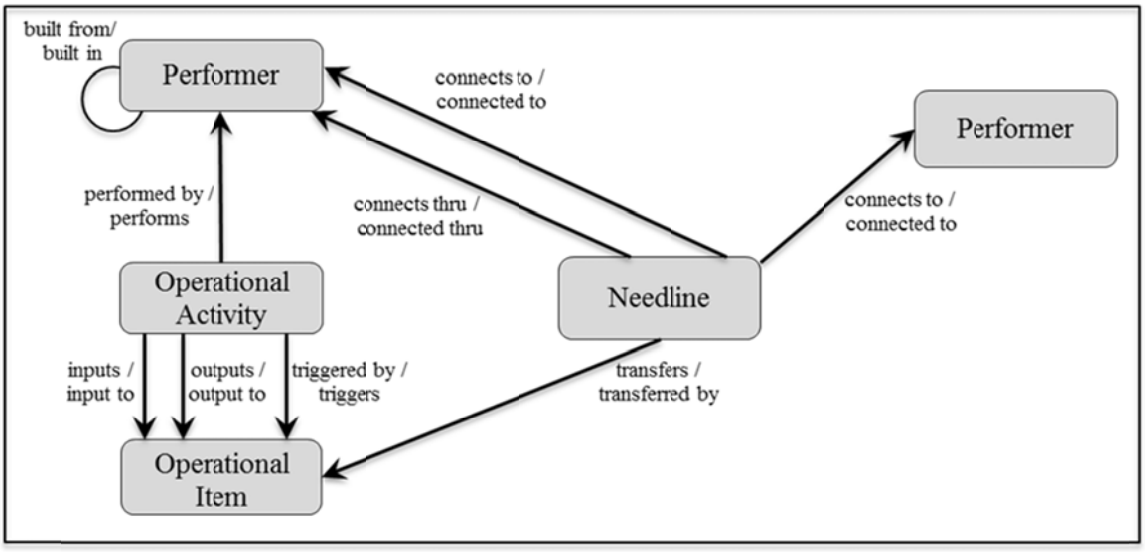


Figure 9 External Needline Definition

Table 7 External Needline Definition

Element Class	Attributes	Relationships	Target Classes
Needline	Description Doc. PUID Number	*connects thru / connected thru*[19]	**Performer**
		connects to / connected to	**Performer**
		transfers / transferred by	**OperationalItem**
OperationalItem	See Section 2.1	*transferred by / transfers*	**Needline**
Performer	See Section 3.1	*connected thru / connects thru*[20]	**Needline**
		connected to / connects to	**Needline**

[19] Automatically set based on the operational node hierarchy and *connects to* targets.
[20] Automatically set based on the operational node hierarchy and *connected to* targets.

3.2.1 Derive or Refine Internal Needlines

Within the **Performer** hierarchy, the assignment of **OperationalActivities** to **Performers** establishes the internal **Needlines** of the **Architecture** based on the **OperationalItems** that flow between the assigned **OperationalActivities**. The internal **Needlines** are formalized in the database using the **Needline** element class.

As the **Performer** hierarchy evolves further, the terminus point for **Needlines** are appropriately changed to lower-level **Performers** where the **OperationalActivities** *performed by* that **Performer** provide the **OperationalItems** transferred by the **Needlines**. When the target of a *connects to* relationship is changed from the **Performer** to one of its subordinates, CORE automatically establishes the *connected thru* relationship between the **Needline** and the parent of the subordinate **Performer**. This allows **Needlines** to retain their identity even though their end points may change as the **Performer** hierarchy grows in depth.

Needlines may be *specified by* performance and constraint **Requirements**. Only the lowest layer of **OperationalItem** should be *transferred by* a **Needline**.

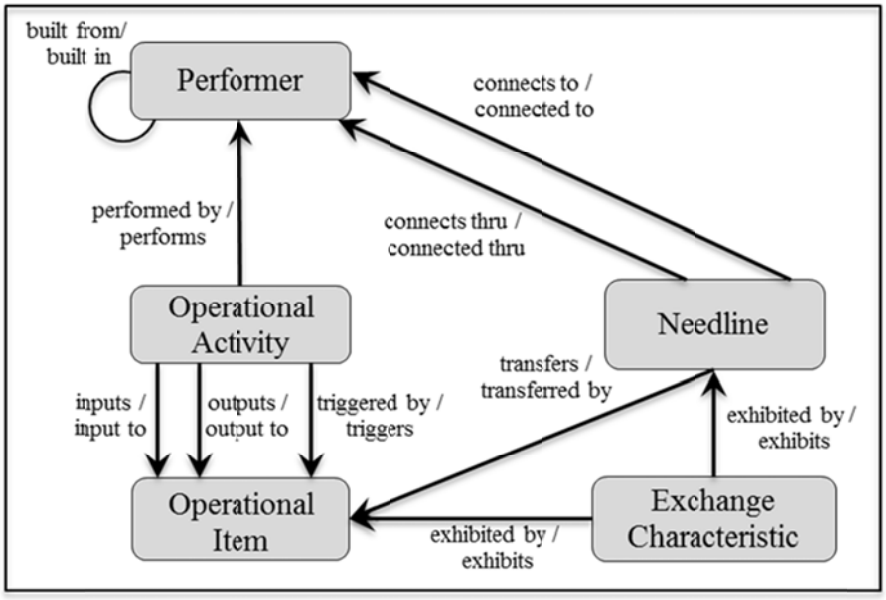


Figure 10 Internal Needline Definition

Table 8 Internal Needline Definition

Element Class	Attributes	Relationships	Target Classes
ExchangeCharacteristic	Access Control Availability Confidentiality Criticality Description Dissemination Control Doc. PUID Integrity Interoperability Level Non-Repudiation Consumer Non-Repudiation Producer Number Periodicity Protection Duration Protection Type Throughput Transaction Type Triggering Event	*exhibited by / exhibits*	**Needline** **OperationalItem**
Needline	See Section 3.2.1	*connects thru / connected thru*[21]	**Performer**
		connects to / connected to	**Performer**
		exhibits / exhibited by	**ExchangeCharacteristic**
		transfers / transferred by	**OperationalItem**
OperationalItem	See Section 2.1	*exhibits / exhibited by*	**ExchangeCharacteristic**
		transferred by / transfers	**Needline**
Performer	See Section 3.1	*connected thru / connects thru*[22]	**Needline**

[21] Automatically set based on the operational node hierarchy and *connects to* targets.
[22] Automatically set based on the component hierarchy and *connected to* targets.

Table 8 Internal Needline Definition

Element Class	Attributes	Relationships	Target Classes
		connected to / connects to	**Needline**

4 Operational Model Validation Using COREsim

COREsim is a discrete event simulator that executes the operational activity and needline models to provide an assessment of operational architecture performance and to verify the dynamic integrity of the conceptual model. COREsim dynamically interprets a behavior model (i.e., the Enhanced Functional Flow Block Diagram (EFFBD)) in conjunction with the needline model and identifies and displays timing, resource utilization, operational item flow, and model inconsistencies. COREsim usage should be an integral part of operational analysis and operational architecture synthesis.

5 Systems Architecture Considerations

Definition of the systems architecture should be done consistent with the structured approach documented in the SDG. Although the systems architecture may involve numerous systems, the SDG principles remain unchanged. System engineering/architecture activities needed to complete the architecture and to interrelate the operational and systems domains are addressed in the following sections.

5.1 Systems Performance Parameters

Elements in the **PerformanceCharacteristics** class are used to capture performance parameters for system elements. Performance parameters include both current values for existing elements and threshold and objective values per time frame for existing or new elements.

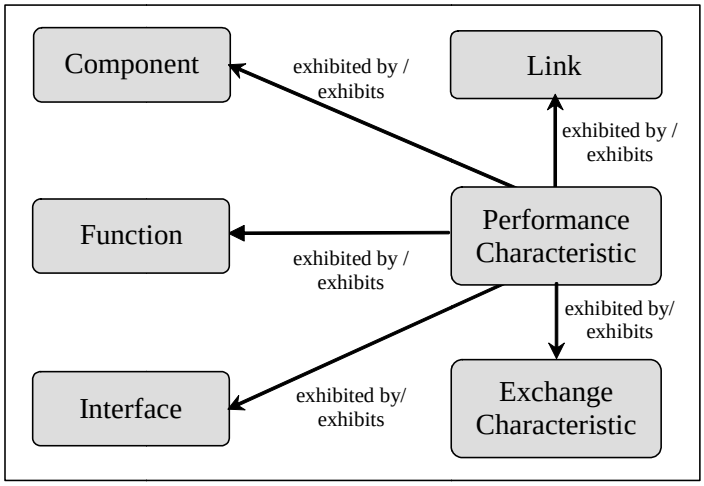


Figure 11 Performance Parameters

Table 9 Performance Parameters

Element Class	Attributes	Relationships	Target Classes
Component	See SDG	*exhibits / exhibited by*	**PerformanceCharacteristic**
ExchangeCharacteristic	See Section 3.2.1	*exhibits / exhibited by*	**PerformanceCharacteristic**
Function	See SDG	*exhibits / exhibited by*	**PerformanceCharacteristic**
Interface	See SDG	*exhibits / exhibited by*	**PerformanceCharacteristic**
Link	See SDG	*exhibits / exhibited by*	**PerformanceCharacteristic**
PerformanceCharacteristic	Current Value Description Number Objective[23] Threshold Time Frame Units	*exhibited by / exhibits*	**Component** **ExchangeCharacteristic** **Function** **Interface** **Link**

5.2 Services Development

Services exist as both a subset of functional behavior and as part of a system. Within the functional behavior model [in the **Function** class], all leaf-level elements that compose the functionality of a service are collected under a root **Function** via the *decomposed by* relation.

Services are created as a **Component** element with the type attribute set to Service. The Service Type attribute should be set to Consumer, Provider, or Both as appropriate. The **Component** element *performs* the root **Function**, with the *performs* behavior type attribute set to: Integrated (Services).

A service specification contains the attributes of a service to be included in the DoDAF viewpoints for a net-centric environment or hybrid system. Service attributes for an internal service [one which is being developed] are developed throughout the operational and system analysis process and are documented in the **ServiceSpecification** class. Service attributes for an external service [one which is an external in the system context] are provided by the service provider. A **Component** of type: Service is *documented by* a **ServiceSpecification**.

[23] The Threshold, Objective, and Time Frame attributes are ordered collections and, therefore, the Threshold and Objective values should be ordered to correspond to the Time Frame values referenced. For example, if the time frame values are near-term, mid-term, and far-term, then the first entry in the Threshold or Objective attribute should be the corresponding near-term value, the second entry the mid-term value, etc.

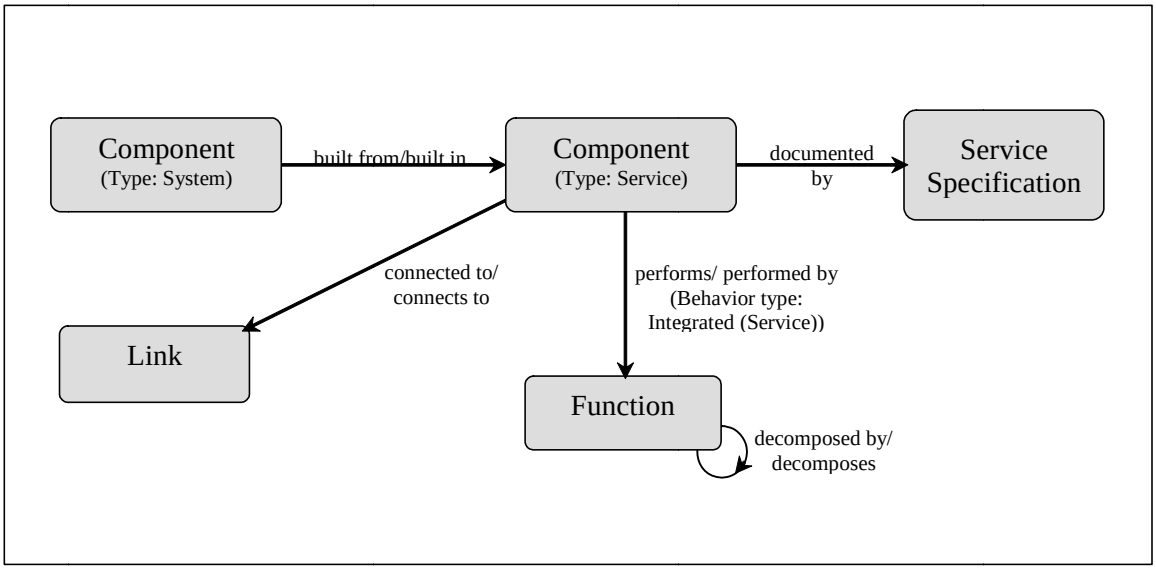


Figure 12 Services

Table 10 Services

Element Class	Attributes	Relationships	Target Classes
Component	See SDG	*built from/ built in*	**Component** (Type: Service)
Component (Type: Service)	See SDG Type: Service	*joined to/joins*	**Interface**
		performs / performed by (Behavior Type: Integrated (Services)	**Function**
		documented by/ documents	**ServiceSpecification**
Function	See SDG	performed by/ performs (Behavior Type: Integrated (Services)	**Component**
Link	See SDG	*connected to /connects to*	**Component**

Table 10 Services

Element Class	Attributes	Relationships	Target Classes
ServiceSpecification	Access Criteria Authentication Mechanism Data Types Effects Information Security Markings Overview Point Of Contact SAP Type Service Access Point Service Version WDSL	*documents / documented by*	**Component** (Type: Service)

5.3 Requirements Development

OperationalActivities and **PerformanceCharacteristics** serve as sources for system **Requirements**. **OperationalActivities** lead to the identification and definition of functional **Requirements**. **PerformanceCharacteristics** lead to the identification and definition of performance **Requirements**. The *results in / result of* relationships are used to map elements in these classes. Thus, a **Requirement** is the *result of* an **OperationalActivity** or a **PerformanceCharacteristic**. See the SDG for a description and use of **Requirement** attributes.

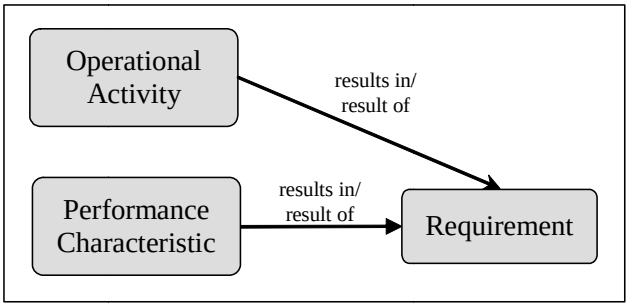


Figure 13 Requirements Development

Table 11 Requirements Development

Element Class	Attributes	Relationships	Target Classes
OperationalActivity	See Section 2.1	*results in / result of*	**Requirement**
PerformanceCharacteristic	See Section 5.1	*results in / result of*	**Requirement**

Table 11 Requirements Development

Element Class	Attributes	Relationships	Target Classes
Requirement	Description Doc. PUID Key Performance Parameter Number Origin: Originating Rationale Type Units Value Weight Factor	*result of / results in*	**OperationalActivity** **PerformanceCharacteristic**

5.4 Traceability from Operational Architecture

The *implemented by / implements* relationships map the operational behavior and performers to the system behavioral and physical elements. These relationship pairs enable full traceability from the operational domain into either the system's physical domain, functional domain or both and, therefore, make it easier for the system engineering team to assess the impacts in the system domain when changes occur within the operational domain. Conversely, the reverse mapping of the system domain into the performers, operational behavior, or both again makes it easier for the system engineering/architecture team to assess the impacts within the operational domain when changes occur in the systems domain. See the SDG regarding **Component**, **Function**, **Item,** and **Link**.

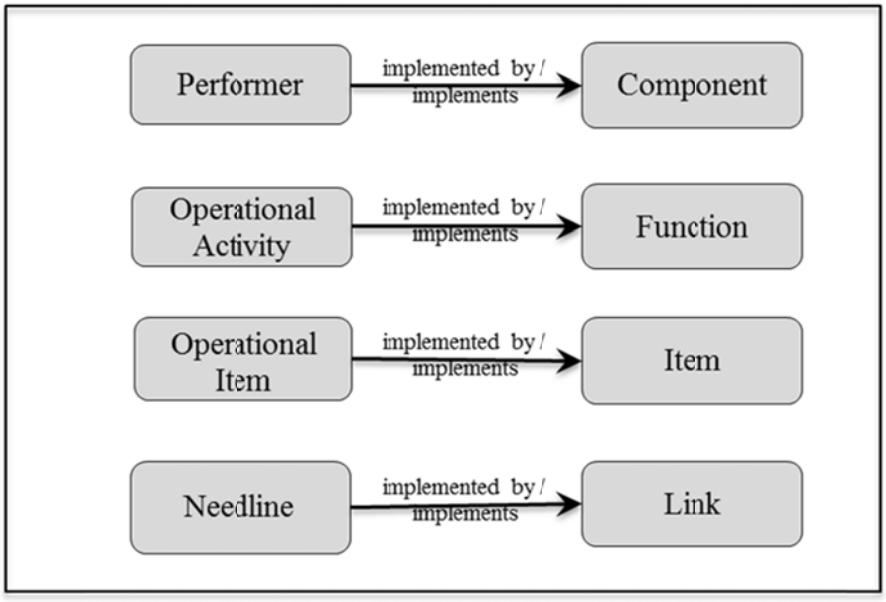


Figure 14 Operational to Systems Traceability

Table 12 Operational to Systems Traceability

Element Class	Attributes	Relationships	Target Classes
Component	See SDG	*implements / implemented by*	**Performer**
Function	See SDG	*implements / implemented by* (Status: nil, Planned, Partial, or Full)	**OperationalActivity**
Item	See SDG	*implements / implemented by*	**OperationalItem**
Link	See SDG	*implements / implemented by*	**Needline**
Needline	See Section 3.2.1	*implemented by / implements*	**Link**
OperationalActivity	See Section 2.1	*implemented by / implements* (Status: nil, Planned, Partial, or Full)	**Function**
OperationalItem	See Section 2.1	*implemented by / implements*	**Item**

Table 12 Operational to Systems Traceability

Element Class	Attributes	Relationships	Target Classes
Performer	See Section 1.4	*implemented by / implements*	**Component**

6 Program Management Aspects

Managing architecture development and systems development within a MBSE environment should conform to whether the programs or projects are top-down, bottom-up, or middle-out in nature. The DoDAF-described Models within the Project Viewpoint describe how programs, projects, portfolios, or initiatives deliver capabilities, the organizations contributing to them, and dependencies among them. Previous versions of DoDAF took a traditional modeling approach of architecture in which descriptions of programs and projects were considered outside DoDAF's scope. To compensate for this, various DoDAF views represented the evolution of systems, technologies and standards (e.g., Systems and Services Evolution Description, Systems Technology Forecast, and Technical Standards Forecast), which had a future programmatic cast. The integration of Project Viewpoints (organizational and project-oriented) with the more traditional architecture representations characterizes DoDAF v2.0-based enterprise architectural descriptions.

6.1 Program/Project Basics

Organizations and **Architectures** are related through the Program/Project Model to relate the enterprise's **Goals** with the **Architecture** and those **Organizations** involved. The Program or Project model develops from the **ProgramElement** class. Each element within the **ProgramElement** class represents some aspect of the structure of the program or project. These elements are related through the *included in / includes* relationship pair. When complete, the resulting hierarchical structure represents the *Work Breakdown Structure* for the program or project. The Type attribute identifies whether the program element instance is a Program, Project, Work Package or Task. The top-most program element (Type: Program) *implements* an **Architecture.**[24] *Assigned to* each **ProgramElement** is an **Organization,** which is responsible for some aspect of the program/project.

The top-most **ProgramElement** *achieves* one or more enterprise-level objectives, which are represented as elements within the **Goal** class. **Goals** describe the desired effect (outcome) or achievement level in operational processes, projects, or special programs. **Goals** may also express enterprise objectives—high-level strategic objectives applying to the entire organization—or as more specific operational objectives that define desired outcomes of the work process. Subordinate **goals** may be *achieved by* lower-level **ProgramElements** (Type: Program or Project). Program/Project risks are followed and managed through the **Risk** class. Normally, a **ProgramElement** *resolves* a **Risk** by instituting strategies to mitigate the risk; however, provision is made for those cases where a **ProgramElement** may in itself *cause* a **Risk**, which program managers must mitigate. The acquisition of **Capabilities** is another important aspect of Program Management. A **Capability** is *provided by* a **ProgramElement,** which *implements* an **Architecture**. Note: A **Capability** is the *basis of* an **OperationalActivity** (see Section 2.1).

[24] Enterprise architecture would cover multiple programs and each program may include multiple projects.

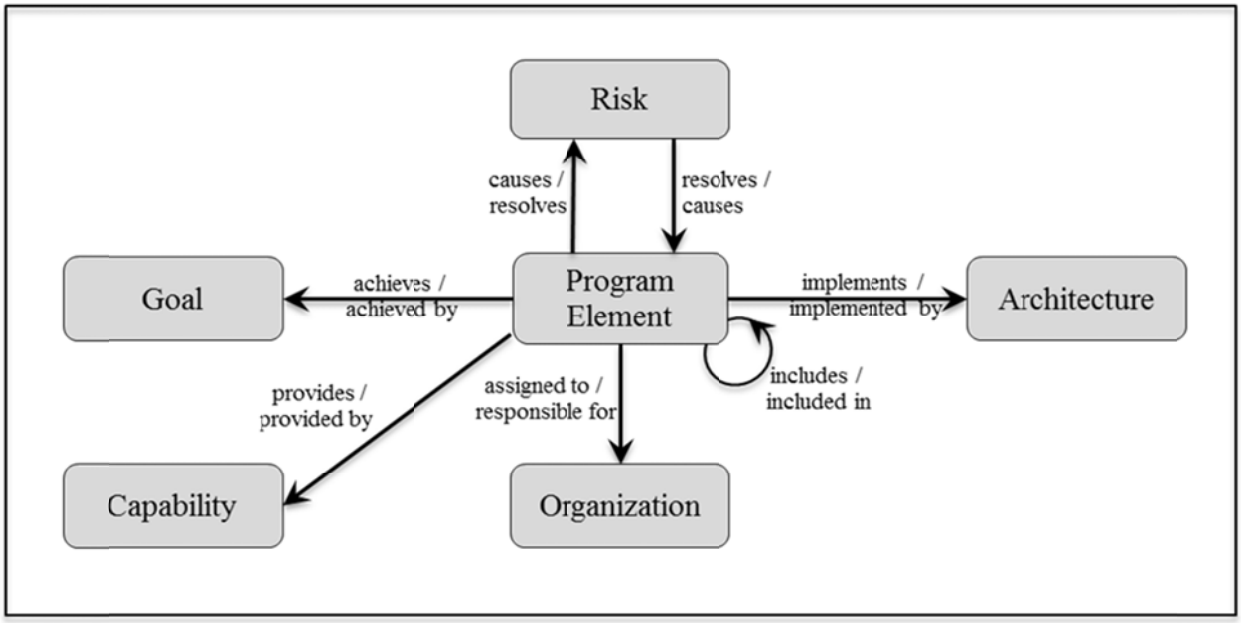


Figure 15 Program Management Basics

Table 13 Program Management Basics

Element Class	Attributes	Relationships	Target Classes
Architecture	See Section 1.1	*implemented by / implements* (Status: nil, Planned, Partial, or Full)	**ProgramElement**
Capability	See Section 1.2	*provided by / provides*	**ProgramElement**
Goal	Description Number	*achieved by / achieves*	**ProgramElement**
		augmented by / augments	**ExternalFile**
		documented by/ documents	**Document**
Organization	See Section 1.3	*responsible for / assigned to*	**ProgramElement**
ProgramElement	Contract Number Cost Description End Date Labor Hours Non-recurring	*accomplishes / accomplished by*	**ProgramActivity**
		achieves / achieved by	**Goal**
		assigned to / responsible for	**Organization**

Table 13 Program Management Basics

Element Class	Attributes	Relationships	Target Classes
	Cost Start Date Type	*augmented by / augments*	**ExternalFile**
		causes / resolves	**Risk**
		implements / implemented by	**Architecture**
		included in / includes	**ProgramElement**
		includes / included in	**ProgramElement**
		provides / provided by	**Capability**
		resolves / causes	**Risk**
		specified by / specifies	**Requirement**
		supplies / supplied by	**Component Performer**

6.2 Program Management Activity Model

Another important facet of program management is developing and maintaining program or project schedules, i.e., timelines. These timelines are established through the **ProgramActivity** class. The **ProgramActivity** class allows the program management team to establish the sequencing of work necessary to accomplish the Task, Work Package, Project or Program of a **ProgramElement**.

The **ProgramActivity** behavior of a **ProgramElement** of Type: Project is the cumulative behaviors of all subordinate **ProgramElement** behaviors. The intent of each **ProgramElement** element is *accomplished by* a **ProgramActivity** and correspondingly, the behavior of each **ProgramActivity** *accomplishes* the intent of its **ProgramElement**. The integrated **ProgramActivity** behavior is developed from integrating subordinate Task, Work Package or Project behaviors (workflows) into a single behavior model that fully represents the workflow required by the parent **ProgramActivity**. COREsim (see Section 4) will execute the program activity models to provide an assessment of the timeline performance (schedule) and to verify the dynamic integrity of the conceptual program management model. COREsim dynamically interprets a behavior model (i.e., the Enhanced Functional Flow Block Diagram (EFFBD)) and identifies and displays timing, resource usage, product flow, and model inconsistencies.

ProgramActivity Inputs and Outputs. Each **ProgramActivity's** integrated behavior will have input and output **Product** elements identified. These **Product** elements are associated with **ProgramActivities** using the relationships: *input to/inputs*, *output from/outputs*, and

triggers/triggered by. As with **ProgramActivities**, **Products** should be aggregated to simplify presentation.

ProgramActivity Traceability. **ProgramActivity** traceability from an appropriate **Goal** element is established using the *achieved by* relationship to an intermediary **ProgramElement**. The **ProgramElement's** *accomplished by* relationship identifies the **ProgramActivities** that apply for accomplishing that **goal's** achievement.

ProgramActivity traceability from an appropriate **Capability** occurs through the *provided by* relationships to an intermediary **ProgramElement**. The **ProgramElement's** *accomplished by* relationship identifies the **ProgramActivities** that apply for accomplishing that **capability's** achievement.

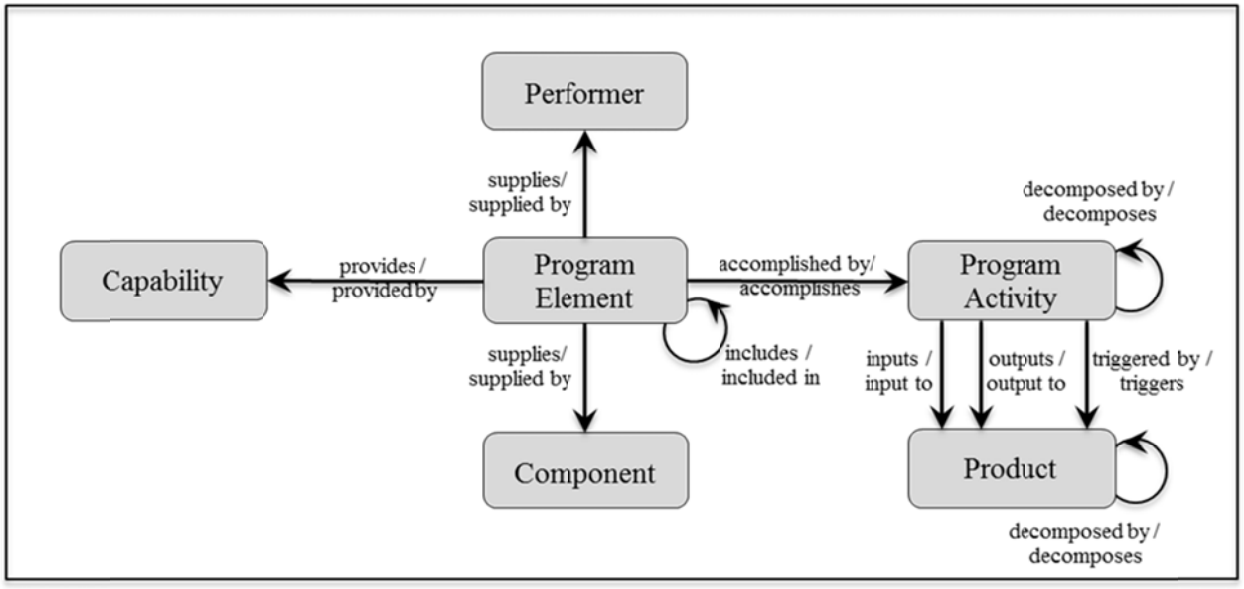


Figure 16 Program Activity Model

Table 14 Program Activity Model

Element Class	Attributes	Relationships	Target Classes
Capability	See Section 1.2	*provided by / provides*	**ProgramElement**
Component	See SDG	*supplied by / supplies*	**ProgramElement**
Performer	See Section 1.1	*supplied by / supplies*	**ProgramElement**
ProgramElement	Contract Number Cost Description End Date Labor Hours Non-recurring	*accomplishes / accomplished by*	**ProgramActivity**
		achieves / achieved by	**Goal**
		assigned to / responsible for	**Organization**

Table 14 Program Activity Model

Element Class	Attributes	Relationships	Target Classes
	Cost Start Date Type	*augmented by / augments*	**ExternalFile**
		causes / resolves	**Risk**
		implements / implemented by	**Architecture**
		included in / includes	**ProgramElement**
		includes / included in	**ProgramElement**
		provides / provided by	**Capability**
		resolves / causes	**Risk**
		specified by / specifies	**Requirement**
		supplies / supplied by	**Component Performer**
Product	Description Number Size Size Units Type	*augmented by / augments*	**ExternalFile**
		decomposed by / decomposes	**Product**
		decomposes / decomposed by	**Product**
		documented by / documents	**Document**
		input to / inputs	**ProgramActivity**
		output from / outputs	**ProgramActivity**
		specified by / specifies	**Requirement**
		triggers / triggered by	**ProgramActivity**

7 Documentation—DoDAF v2.0 Viewpoints

CORE includes a set of scripts to output each of the DoDAF v2.0 viewpoints as Rich Text Format (RTF) documents. As appropriate to the particular viewpoint, each viewpoint document contains a standard CORE diagram, a table generated from the contents of the repository, or an external file referenced by an **ExternalFile** element. Because the viewpoints are generated as a result of applying the model-based system engineering process to architecture definition, these scripts have been designed to be flexible in order to support the architects/system engineers developing the architecture on an on-going basis and to produce the viewpoints for customer usage.

Table 15 DoDAF v2.0 Viewpoint Scripts

Viewpoint	Viewpoint Title	Script Output
AV-1	Overview and Summary Information	User selected **Architecture** Description, Purpose, Scope, Time Frame, *achieves* **Mission** name and Description, and *augmented by* **Text** and **ExternalFiles**.
AV-2	Integrated Dictionary	User selected **Architecture**.
CV-1	Vision	User selected **Architecture** *implemented by* **ProgramElement** which *provides* **Capability.**
CV-2:	Capability Taxonomy	User selected **Architecture** *implemented by* **ProgramElement** which *provides* **Capability** and **Capability** is *refined by* **Capability.**
CV-3	Capability Phasing	User selected **Architecture** *implemented by* **ProgramElement**, which *supplies* **Capabilities** determine when projects providing elements of capability are to be delivered, upgraded and/or withdrawn.
CV-4	Capability Dependencies	**Category** *categorizes* **Capability**
CV-5:	Capability to Organizational Development Mapping	User selected **Architecture specified by Capability refined by Capability**
CV-6	Capability to Operational Activities Mapping	User selected **Architecture** *specified by* **Capability** *refined by* **Capability** *basis of* **OperationalActivity** *performed by* **Performer**
CV-7	Capability to Services Mapping	Matrix mapping **Capability** to **Performer** of type Service Functionality Provider
DIV-1	Conceptual Data Model	Data elements used and their attributes and relations.
DIV-2	Logical Data Model	Outputs characteristics of **OperationalItems** that are *output from*, *input to*, or *triggers* a user selected **OperationalActivity**, its children, and, optionally, their children.

Table 15 DoDAF v2.0 Viewpoint Scripts

Viewpoint	Viewpoint Title	Script Output
DIV-3	Physical Data Model	Outputs a user selected **OperationalItem** characteristics table for **OperationalItems** related to a user selected **OperationalActivities**, its children, and, optionally, their children.
OV-1	High-Level Operational Concept Graphic	User selected **ExternalFile**.
OV-2	Operational Resource Flow Description	Physical Block Diagram (PBD) for user selected **Performer.**
OV-3	Operational Resource Flow Matrix	Summary matrix or full matrix for information exchanges of the children of **OperationalActivity**(s) *performed by* **Performers** that *compose* the user selected **Architecture**.
OV-4	Organization Relationships Chart	Organization Hierarchy for the user selected **Organization.**
OV-5a	Operational Activity Decomposition Tree	Functional Hierarchy for **OperationalActivity**(s) *performed by* **Performers** that *compose* the user selected **Architecture**.
OV-5b	Operational Activity Model	IDEF0 for user selected **OperationalActivity** and, optionally, its children. Includes optional output of Function Hierarchy for selected **OperationalActivity**. Automatically outputs A-0 diagram for selected **OperationalActivity**.
OV-6a	Operational Rules Model	EFFBD or Activity Diagrams for **OperationalActivity(s)** *performed by* **Performers** that *compose* the user selected **Architecture.**
OV-6b	State Transition Description	User selected **ExternalFiles** and **State/Modes** that are *exhibited by* **Performers** that *compose* the user selected **Architecture**.
OV-6c	Event-Trace Description	Sequence Diagrams for **OperationalActivity(s)** *performed by* **Performers** that *compose* the user selected **Architecture**.
PV-1	Project Portfolio Relationships	Item characteristics table for **OperationalItem** linked to user selected **OperationalActivity**, its children, and, optionally, their children.
PV-2	Project Timelines	User selected **ExternalFile**.
PV-3	Project to Capability Mapping	User selected **ProgramElements** mapping to **Capabilities**.
SvcV-1	Services Context	Interface Block Diagram for **Component(s)** type Service that *composes* the user selected

Table 15 DoDAF v2.0 Viewpoint Scripts

Viewpoint	Viewpoint Title	Script Output
	Description	**Architecture.**
SvcV-2	Services Resource Flow Description	Physical Block Diagram for **Component(s)** type Service that *composes* the user selected **Architecture.**
SvcV-3a	Systems-Services Matrix	Matrix indentifying interfaces between children of **Component(s)** type Service that *composes* the user selected **Architecture** and **Component(s)** type System.
SvcV-3b	Services-Services Matrix	Matrix indentifying interfaces between children of **Component(s)** type Service that *composes* the user selected **Architecture.**
SvcV-4	Services Functionality Description	IDEF0 diagrams for **Function(s)** *performed by* **Component(s)** type Service that *composes* the user selected **Architecture.**
SvcV-5	Operational Activity to Services Traceability Matrix	Matrix mapping **Functions** *performed by* **Component(s)** type Service that *composes* the user selected and their associated **Interfaces**, **Links**, and **Functions** to **OperationalActivity(s).**
SvcV-6	Services Resource Flow Matrix	Summary matrix or full matrix for data exchanges of the children of **Component (s)** type Service that *compose*s the user selected **Architecture**.
SvcV-7	Services Measures Matrix	Quantitative characteristics for the children of **Component(s)** type Service that composes the user selected and their associated **Interfaces, Links,** and **Functions**. Contains both the current **PerformanceCharacteristics** as well as the expected or required performance parameters.
SvcV-8	Services Evolution Description	User selected **ExternalFile**.
SvcV-9	Services Technology & Skills Forecast	User selected **ExternalFile**.
SvcV-10a	Services Rules Model	EFFBD or Activity Diagram diagrams for **Function(s)** *performed by* **Component(s)** type Service that *composes* the user selected **Architecture.**
SvcV-10b	Services State Transition Description	User selected **ExternalFiles** and **State/Modes** that are *exhibited by* **Component(s)** type Service that *composes* the user selected **Architecture**.
SvcV-10c	Services Event-Trace Description	Sequence Diagrams for **Functions** *performed by* **Component(s)** type Service that *composes* the user

Table 15 DoDAF v2.0 Viewpoint Scripts

Viewpoint	Viewpoint Title	Script Output
		selected **Architecture**.
StdV-1	Standards Profile	A listing of standards that apply to solution elements along with the description of emerging standards and potential impact on current solution elements, within a set of time frames.
StdV-2	Standards Forecast	See StdV-1
SV-1	Systems Interface Description	Interface Block Diagram for **Component(s)** type System that *composes* user selected **Architecture**.
SV-2	Systems Resource Flow Description	Physical Block Diagram for **Component(s)** type System that *composes* user selected **Architecture**.
SV-3	Systems-Systems Matrix	Matrix indentifying interfaces between children of **Component(s)** type System that *composes* the user selected **Architecture**.
SV-4	Systems Functionality Description	IDEF0 diagrams for **Function(s)** *performed by* **Component(s)** type System that *composes* the user selected **Architecture**.
SV-5a	Operational Activity to Systems Function Traceability Matrix	Matrix mapping Functions *performed by* **Component(s)** type System that *composes* the user selected **Architecture** and their associated **OperationalActivity(s).**
SV-5b	Operational Activity to Systems Traceability Matrix	Matrix mapping **Component(s)** type System that *composes* the user selected **Architecture** and their associated **OperationalActivity(s).**
SV-6	Systems Resource Flow Matrix	Summary matrix or full matrix for data exchanges of the children of **Component(s)** type System that composes the user selected **Architecture.**
SV-7	Systems Measures Matrix	Quantitative characteristics for the children of the user selected **Component** and their associated **Interfaces**, **Links**, and **Functions**. Contains both the current performance characteristics as well as the expected or required performance parameters.
SV-8	Systems Evolution Description	User selected **ExternalFile**.
SV-9	Systems Technology & Skills Forecast	User selected **ExternalFile**.
SV-10a	Systems Rules Model	EFFBD or Activity diagrams for **Function(s)** performed by **Components(s)** type System that composes the user selected **Architecture.**

Table 15 DoDAF v2.0 Viewpoint Scripts

Viewpoint	Viewpoint Title	Script Output
SV-10b	Systems State Transition Description	User selected **ExternalFiles** and **State/Modes** that are exhibited by **Component(s)** type System that *composes* the user selected **Architecture**.
SV-10c	Systems Event-Trace Description	Sequence Diagrams for **Fucntion(s)** *performed by* **Component(s)** type System that *composes* the user selected **Architecture**.

In addition to the DoDAF viewpoint scripts, CORE provides numerous engineering support scripts such as the Generic Table Output, Indented Hierarchy Reports, Element Definition, HTML Report, et al. These should be used on an on-going basis to aid the system engineers in communication and assessment of the architecture definition.

Vitech Corporation

2270 Kraft Drive, Suite 1600
Blacksburg, Virginia 24060
540.951.3322 FAX: 540.951.8222
Customer Support: support@vitechcorp.com
www.vitechcorp.com