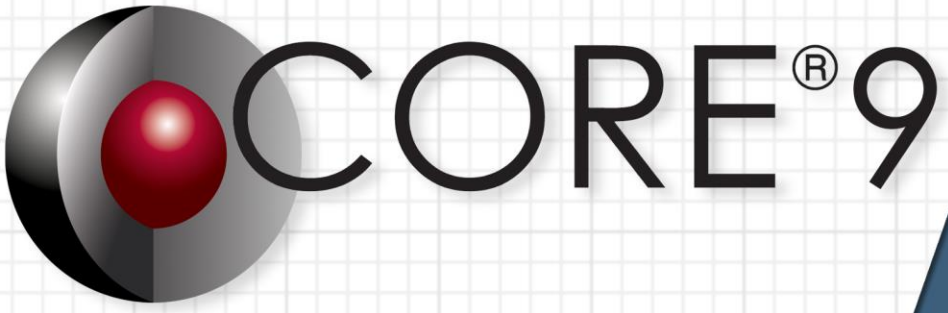

Architecture Definition Guide

Vitech

Copyright © 1998-2013 Vitech Corporation. All rights reserved.

No part of this document may be reproduced in any form, including, but not limited to, photocopying, translating into another language, or storage in a data retrieval system, without prior written consent of Vitech Corporation.

Restricted Rights Legend

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.277-7013.

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

CORE® is a registered trademark of Vitech Corporation.

Other product names mentioned herein are used for identification purposes only, and may be trademarks of their respective companies.

Publication Date: June 2013

TABLE OF CONTENTS

1. Preface v
2. Architecture Concepts viii
   2.1 Operational and System Architecture Domain Relationships ix
3. Operational Concept Capture 1
   3.1 Define Architecture 1
   3.2 Capture Source Material 2
   3.3 Identify Organizations 6
   3.4 Define Operational Boundary 7
   3.5 Classification 8
4. Operational Activity Analysis 9
   4.1 Operational Activity View 9
   4.2 State View 12
5. Operational Architecture Synthesis 15
   5.1 Assign OperationalActivities to Next Level of Performers 15
   5.2 Refine External Needline Definitions 16
   5.3 Derive or Refine Internal Needlines 18
6. Operational Viewpoint Validation Using COREsim 19
7. Operational Architecture Considerations 19
   7.1 Performance Requirements 19
   7.2 Services Development 20
   7.3 Requirements Development 21
   7.4 Traceability from Operational Architecture 22
8. Program Management Aspects 23
   8.1 Program/Project Basics 23
   8.2 Program Management Activity View 25
9. Documentation—DoDAF v2.02 Viewpoints 27

LIST OF FIGURES

Figure 1 MBSE Activities vi
Figure 2 CORE's DoDAF v2.02 Schema – Part 1 viii
Figure 3 CORE's DoDAF v2.02 Schema – Part 2 ix
Figure 4 Architecture Definition 1
Figure 5 Source Material 3
Figure 6 Organizations 6
Figure 7 Operational Boundary 7
Figure 8 Classification 8
Figure 9 Operational Activity View 11
Figure 10 State View 13
Figure 11 Performer Hierarchy and OperationalActivity Assignment 16
Figure 12 External Needline Definition 17
Figure 13 Internal Needline Definitions 18
Figure 14 Performance Requirements 19
Figure 15 Services 20
Figure 16 Requirements Development 21
Figure 17 Operational to Systems Traceability 22
Figure 18 Program Management Basics 24
Figure 19 Program Activity View 26

LIST OF TABLES

Table 1 Architecture Definition 1
Table 2 Source Material 3
Table 3 Organizations 7
Table 4 Operational Boundary 7
Table 5 Classification 9
Table 6 Operational Activity View 11
Table 7 State View 14
Table 8 Performer Hierarchy and OperationalActivity Assignment 16
Table 9 External Needline Definition 17
Table 10 Internal Needline Definitions 18
Table 11 Performance Requirements 19
Table 12 Services 20
Table 13 Requirements Development 22
Table 14 Operational to Systems Traceability 22
Table 15 Program Management Basics 24
Table 16 Program Activity View 26
Table 17 DoDAF v2.02 Viewpoint Scripts 27

1. 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 DoDAF, refer to the Department of Defense Architecture Framework Version 2.02, 28 May 2010 (Volume 1, Volume 2, and Volume 3). This guide is written as a supplement to the CORE System Definition Guide (SDG).

A DoDAF architecture contains operational elements, system elements, and program management elements; therefore, enterprise and operational development must consider these three areas.[1] This ADG presents the activities required to capture and develop an operational architecture. Operational viewpoints are developed using model-based systems engineering (MBSE) principles, which apply equally well to architecture development, and the engineering activities. Integration of the operational viewpoints and the system viewpoints occur through the MBSE model as captured in the CORE repository. These architectural developmental activities may be expressed in terms of systems engineering domain activities without loss of specificity or generality. The systems 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 v2.02 viewpoints become available as a consequence of applying MBSE to a specific operational architecture.

This guide describes each architectural development activity and the CORE DoDAF v20 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 repository in a manner that facilitates the production of DoDAF v2.02 viewpoints[2] using the standard scripts provided with CORE.

CORE's Model Assistant, when enabled, offers autogeneration of some class elements, their key relationships, and where used, attributes on those relationships. One aspect introduced with CORE 9 is pre-populating the repository with elements when the project is created. For example, when a project is created, Model Assistant, when enabled, creates a System Context element in the Component class along with a root Function (_Perform System Context Functions) and a root State (_Exhibit System Context States). An additional autogenerated element within the DoDAF schema is "_Architecture Context" in the Performer class.

This guide augments the SDG and the MBSE 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] Enterprise architectures follow these same principles; however, enterprise architectures are not specifically addressed in this architecture definition guide.

[2] 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 2.02 models is possible.

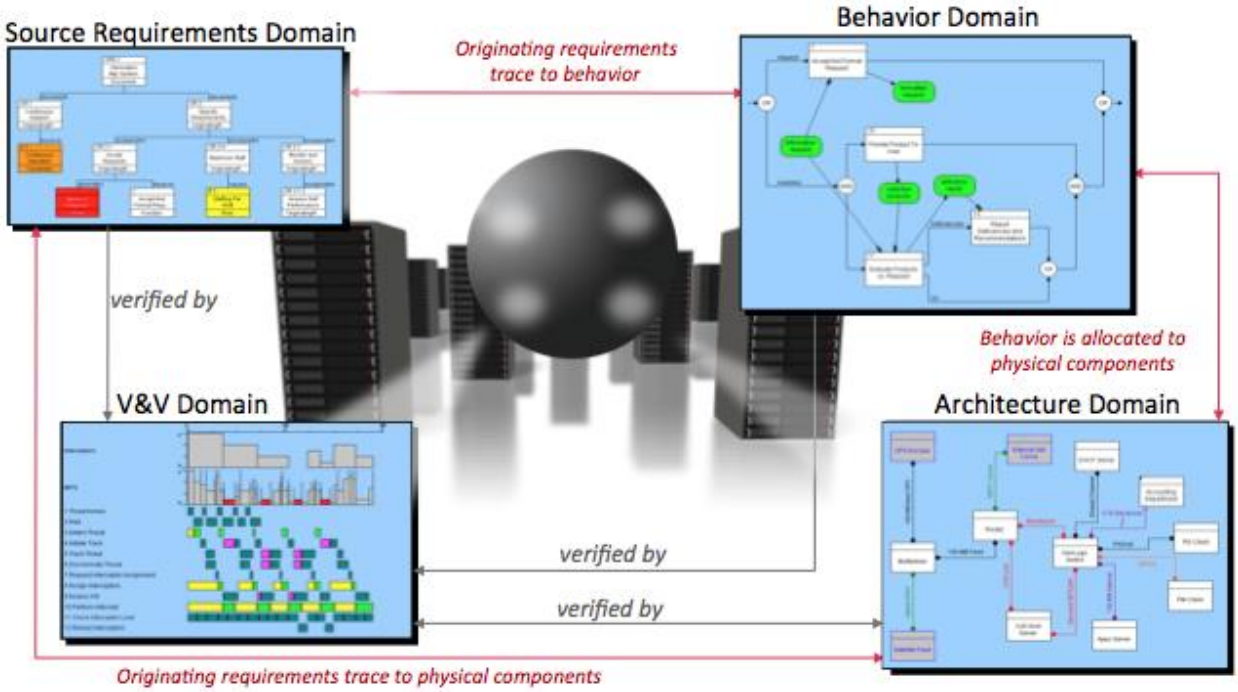

Figure 1 MBSE Activities

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 help documentation.
- For the definition of schema terms, the reader is referred to the CORE DoDAF v20schema, 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 v2.02 viewpoints, the reader is referred to the script help file provided for each DoDAF v2.02 script. The user may access this documentation by selecting the Run Script icon on the CORE toolbar, selecting the DoDAF v20 folder, selecting any one of the DoDAF v2.02 scripts such as the (AV-1) Overview and Summary Information script, and clicking 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.

2. ARCHITECTURE CONCEPTS

As portrayed in Figure 2 CORE's DoDAF v2.02 Schema – Part 1, CORE separates architecture into an Operational Architecture Domain and a System Architecture Domain. The Operational Architecture Domain captures originating concepts, capabilities, and through supporting operational analysis, exposes requirements leading to, and implemented in, the System Architecture Domain.

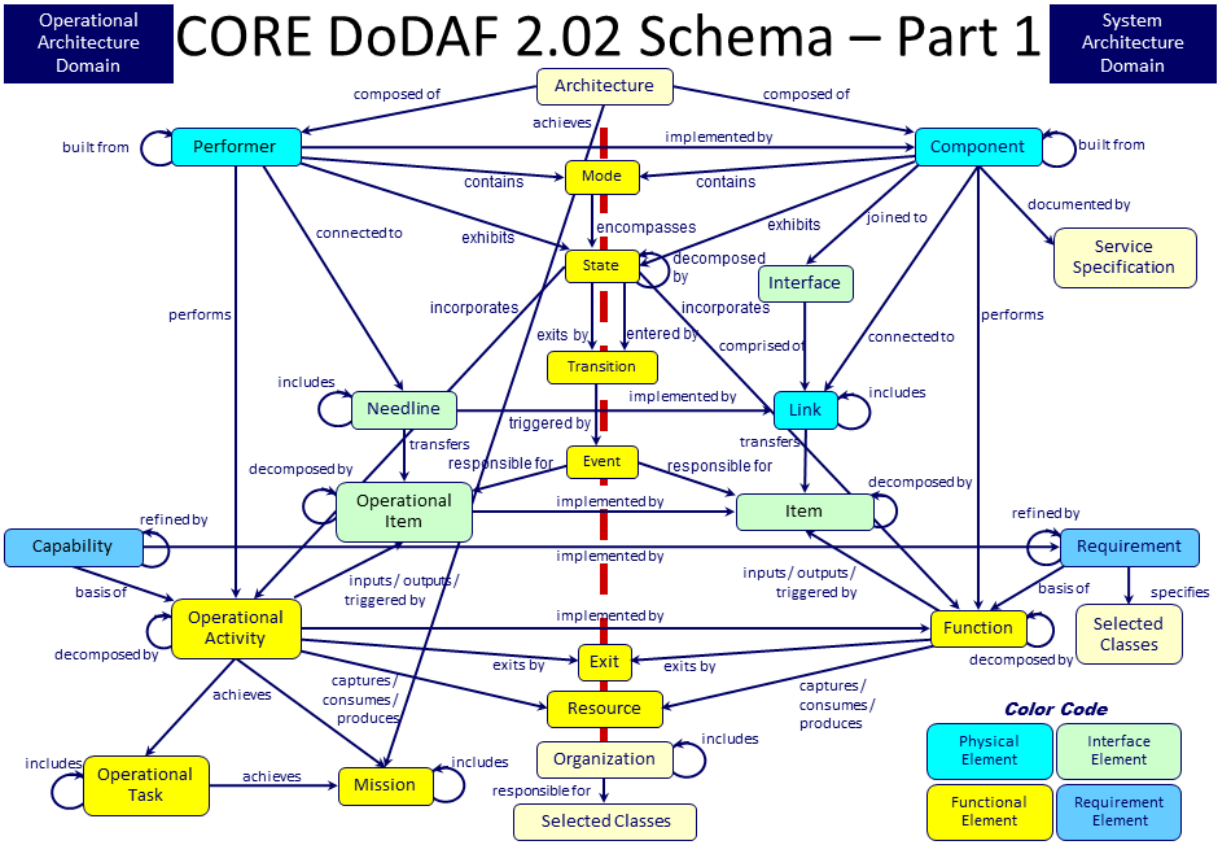

Figure 2 CORE's DoDAF v2.02 Schema – Part 1

As portrayed in Figure 2, CORE's DoDAF v2.02 Schema – Part 2, CORE integrates the Program Management Domain with both the Operational Architecture Domain and the 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.

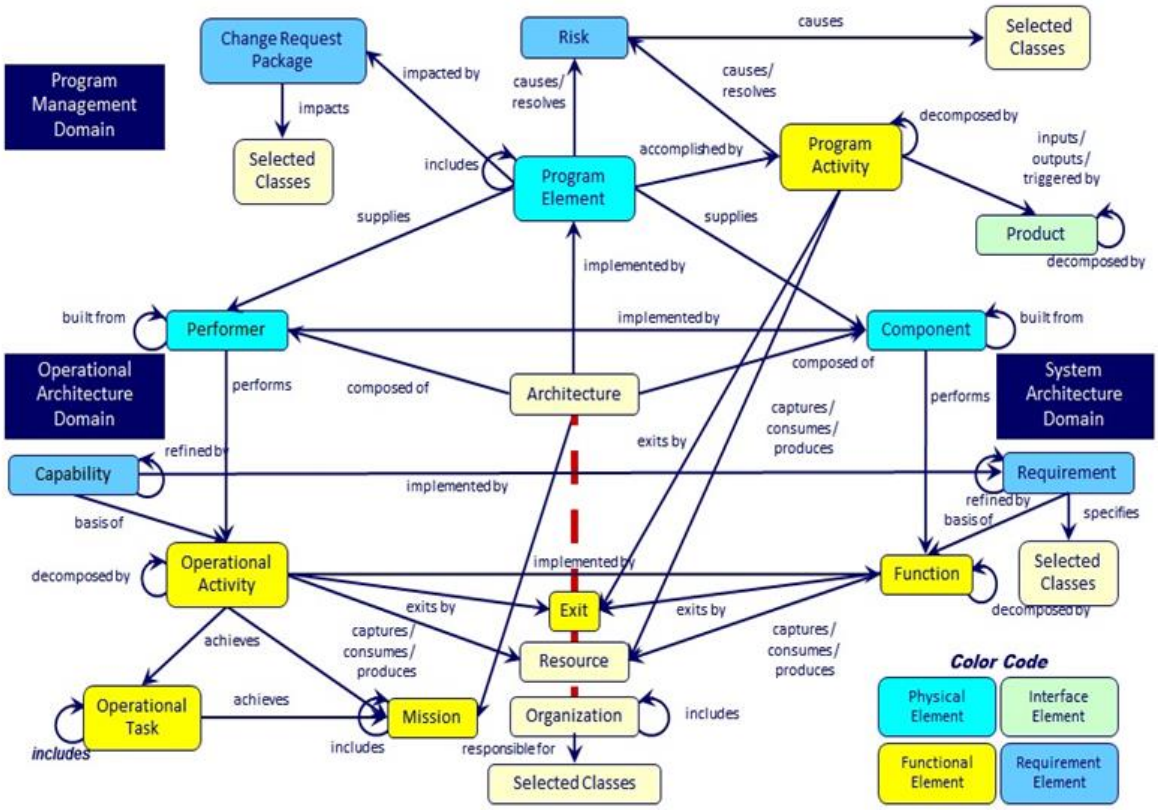

Figure 3 CORE's DoDAF v2.02 Schema – Part 2

This ADG (DoDAF v2.02) provides guidance into structuring the elements, attributes and relationships that implement the Operational Architecture Domain and Program Management Domain for a project. Similarly, the SDG provides guidance into structuring the elements, attributes and relationships that implement the System Architecture Domain.

2.1 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 analyses to support defining 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 Section 3.1 *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 Section 3.4 *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, Section 1.3, Define System Boundary* for details on defining the system boundary.

3. 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 architecture capabilities or requirements that are treated as originating operational requirements and/or architectural guidance information (See Section 3.2 Capture Source Material).

3.1 Define Architecture

Identify the architecture. Architectures exist for the purpose of achieving a well-defined system or more broadly for the enterprise, an integrated set of 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.

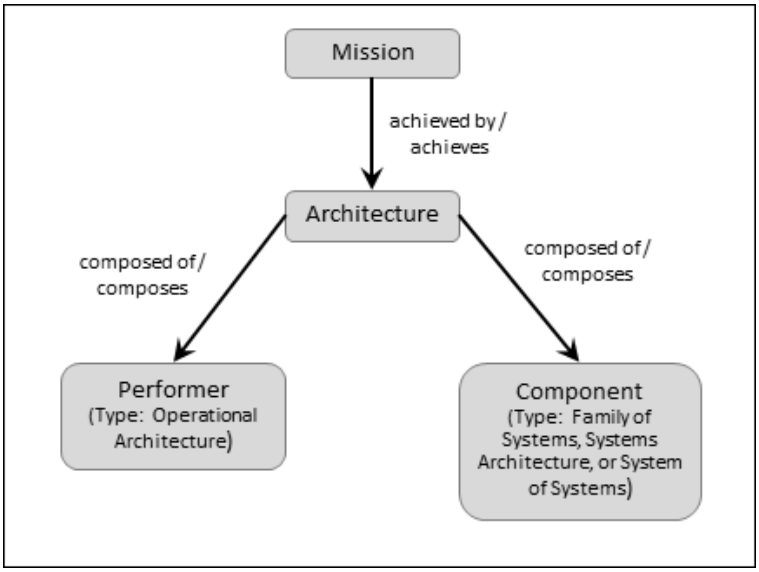

Figure 4 Architecture Definition[3]

Table 1 Architecture Definition

Element Class	Attributes	Relationships	Target Classes
Architecture	Description Number	*composed of / composes*	**Component** **Performer**

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

Table 1 Architecture Definition

Element Class	Attributes	Relationships	Target Classes
	Purpose Scope Time Frame[4]	*achieves / achieved by*	**Mission**
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[5] Longitude Number Operations Purpose Receptions Values Type: Operational Architecture	*composes / composed of*	**Architecture**

3.2 Capture Source Material

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

- **Capability** element for each source capability statement[6]
- **Document** element for each source document
- **Mission** element for each pertinent mission area or description
- **OperationalTask** element for each operational task from a source such as the Universal Joint Task List (UJTL) or the Mission Essential Task List (METL)[7]
- **Requirement** element for each source requirement[8]
- **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 repository using **ExternalFile** elements where each *augments* the subject element. The formal documentation scripts, as well as the System Description Document (SDD) script, will automatically include

[4] It is recommended that the **Architecture** for each distinct time frame be captured in separate CORE Projects.
[5] 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.
[6] A *Capability Requirement* is distinguished from a *Capability* and is placed in the **Requirement** class with the Type attribute set to *Capability.*
[7] 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.
[8] Examples are architecture and operational constraints, task performance characterization, and guidance derived.

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 **ExternalFile**.

- Acronyms and/or special terms appearing in the source document should be captured in the repository 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 SDD.

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

- Document Parser
- Element Extractor
- DOORS Connector tool if the elements are being transferred from IBM® Rational® DOORS®
- CSV Parser script if the elements are being transferred as a CSV file from another application such as Microsoft Excel or Microsoft Access
- Copy and Paste or Paste Unformatted commands

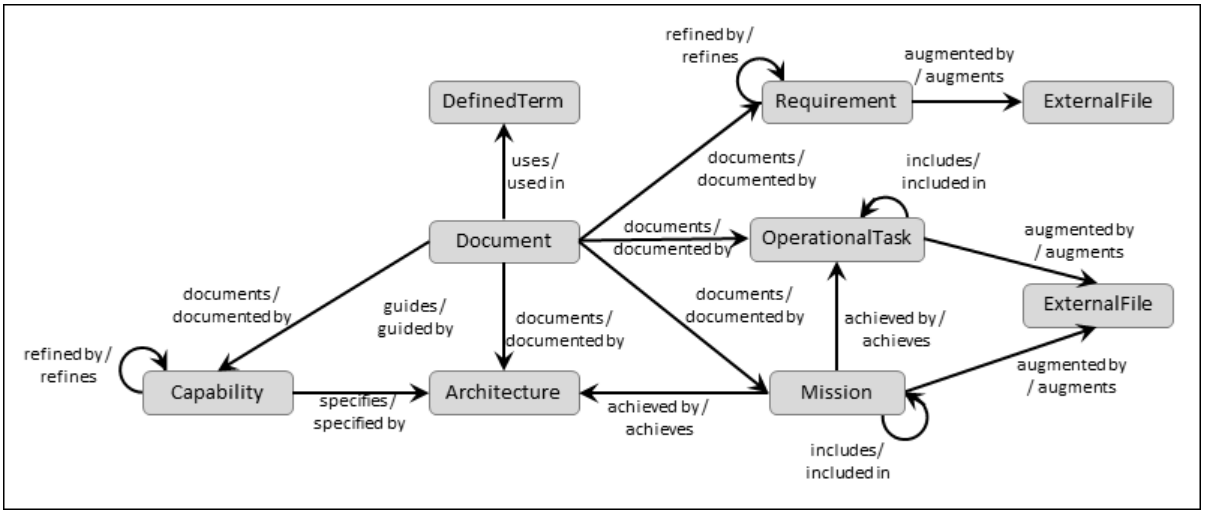

Figure 5 Source Material

Table 2 Source Material

Element Class	Attributes	Relationships	Target Classes
Architecture	See Section 3.1	*achieves / achieved by*	**Mission**
		assigned to / responsible for	**Organization**
		augmented by / augments	**ExternalFile**
		composed of / composes	**Component Performer**
		documented by / documents	**Document**
		implemented by / implements	**ProgramElement**

Table 2 Source Material

Element Class	Attributes	Relationships	Target Classes
		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**
		implemented by / implements	**Requirement**
		refined by / refines	**Capability**
		specified by /	*Requirement*
		specifies / specified by	*Architecture* *Interface* *Needline* *OperationalItem* *Performer* *State*
		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 Revision Number Type	*documents / documented by*[9]	**Architecture** **Capability** **Mission** **Needline** **OperationalActivity** **OperationalItem** **OperationalTask** **Performer** **Requirement** **State**
		uses / used in	**DefinedTerm**

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

Table 2 Source Material

Element Class	Attributes	Relationships	Target Classes
ExternalFile	Description External File Path Number Page Orientation Title Type	*augments / augmented by*[10]	**Architecture** **Capability** **Event** **Mission** **Mode** **Needline** **OperationalActivity** **OperationalItem** **OperationalTask** **Performer** **Requirement** **State** **Transition** **UseCase**
Mission	Description Number	*achieved by / achieves*	**Architecture** **OperationalActivity** **OperationalTask**
		assigned to / responsible for	**Organization**
		augmented by / augments	**ExternalFile**
		documented by / documents	**Document**
		included in / includes	**Mission**
OperationalTask	Description Number	*achieves / achieved by*	**Mission**
		augmented by / augments	**ExternalFile**
		documented by / documents	**Document**
		included in / includes	**OperationalTask**
Requirement	Description Doc. PUID Key Performance Parameter	*augmented by / augments*	**ExternalFile**
		documented by / documents	**Document**

[10] 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.

Table 2 Source Material

Element Class	Attributes	Relationships	Target Classes
	Incentive Performance Parameter[11] Number Origin: Operational Paragraph Number Paragraph Title Rationale Units Value Weight Factor	*refined by / refines*	**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). Consequently, in order to produce professional-looking documents, care should be taken when capturing external source material. Specifically, when using the Element Extractor tool, 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.

3.3 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.

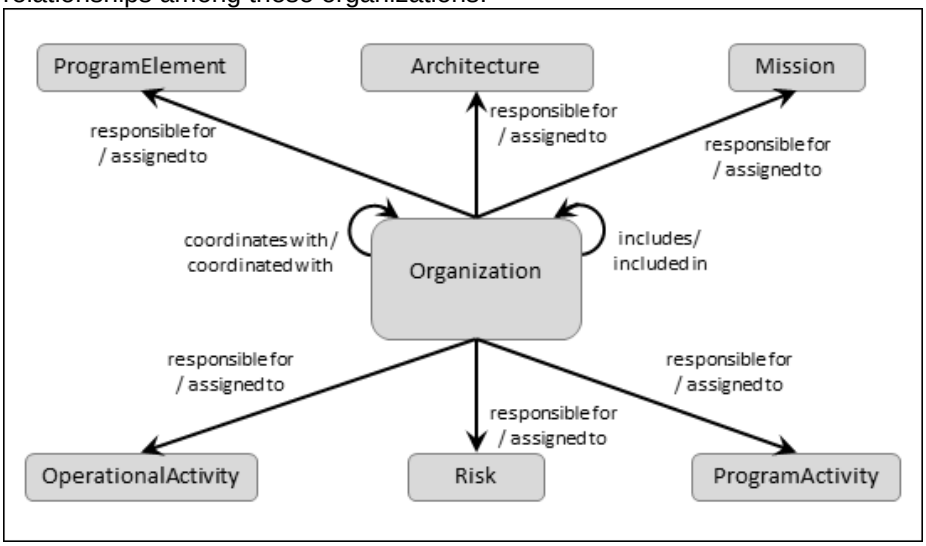

Figure 6 Organizations

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

Table 3 Organizations

Element Class	Attributes	Relationships	Target Classes
Organization	Abbreviation Description Latitude Location Longitude Number Role	*coordinated with / coordinates with*	**Organization**
		included in / includes	**Organization**
		responsible for / assigned to	**Architecture** **OperationalActivity** **Mission** **ProgramActivity** **ProgramElement** **Risk**

3.4 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 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 performer.

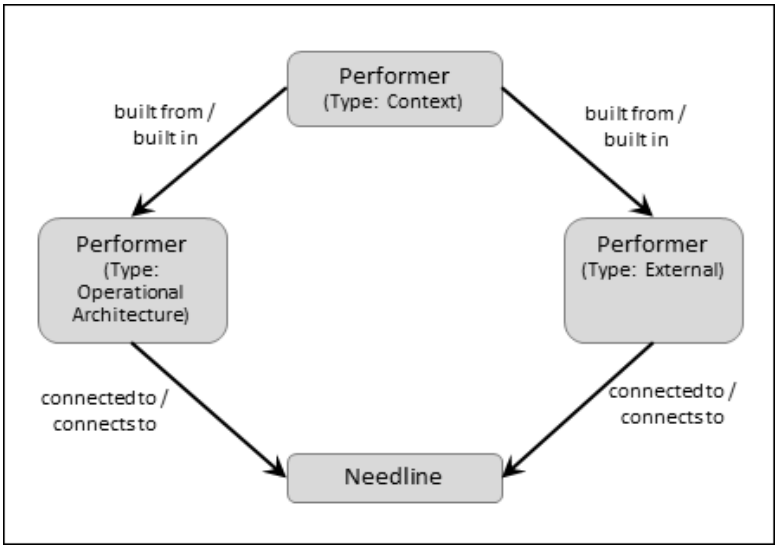

Figure 7 Operational Boundary

Table 4 Operational Boundary

Element Class	Attributes	Relationships	Target Classes
Performer (Type: Context)	See Section 3.1	*built from / built in*	**Performer** (Type: Operational Architecture and

			External)
Performer (Type: External)	See Section 3.1	*built in / built from*	**Performer**
		connected to / connects to	**Needline**
Performer (Type: Operational Architecture)	See Section 3.1	*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.

3.5 Classification

Some architectures must address the classification of objects. The **Classification** class serves to associated classification level and other characteristics, such as Dissemination Control with, potentially, all elements in the repository.

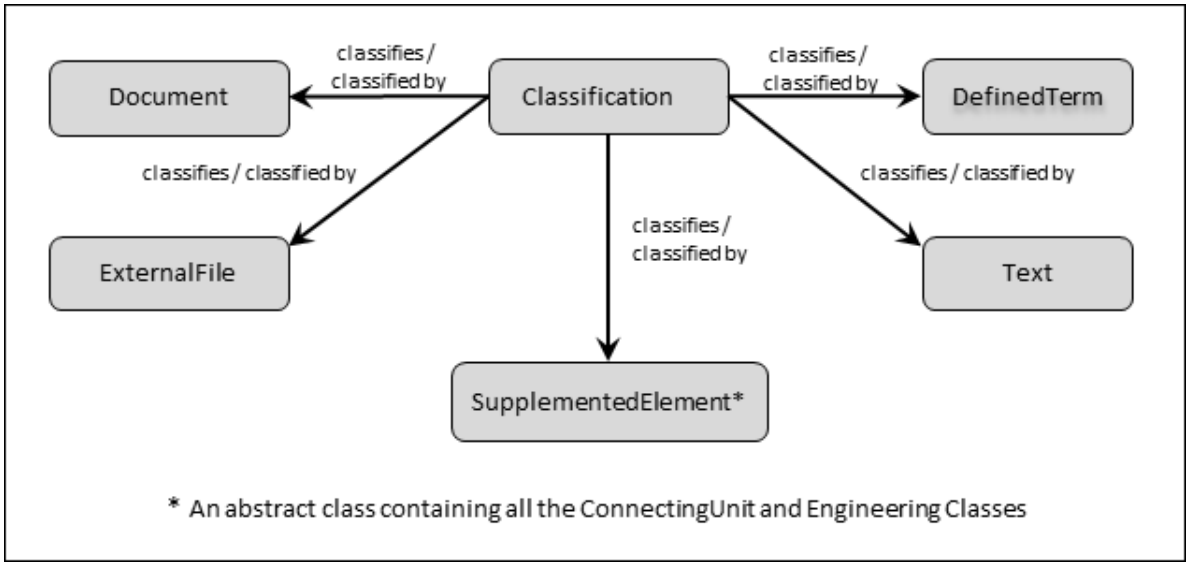

Figure 8 Classification

Table 5 Classification

Element Class	Attributes	Relationships	Target Classes
Classification	Description Number Classification Category Dissemination Control Releasability Security Level Short Label	*classifies / classified by*	**DefinedTerm** **Document** **ExternalFile** **SupplementedElement** **Text**

4. OPERATIONAL ACTIVITY ANALYSIS

Given the need to satisfy an operational mission(s) within the context of a CONOPS and/or an operational requirements document, the systems engineering/architecture team must derive the operational architecture's necessary operational behavior 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 view for the architecture.

4.1 Operational Activity View

Capabilities[12] 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.

The above definition self-interprets "ways and means" as "activities and resources." In MBSE, "ways" are behaviorally interpreted, i.e., Functions, **OpererationalActivities**, etc. "Resources" have a two-fold interpretation. The DoDAF literature predominately sees "resources" as inputs and outputs of **Functions** and **OperationalActivities.** Hence, "resources" are seen primarily as **Items** and **OperationalItems**.

Another usage of "resource" sees it as a necessary object for a behavioral entity to execute, as expected, in a dynamic environment. To illustrate this concept consider a maintenance action, where the function is to transform an inoperative component into a properly functioning component. The functional transform is "repair," where the input is a non-functioning component and the output is a functioning component. An input would obviously be a non-functioning component and an output would obviously be a functioning component. However, notice this "repair" function needs a set of spare parts and other material or non-material to enable the repair function to happen. Lack of spare components causes the repair function to lengthen in its execution.

The language of DoDAF tends to treat these spare components also as **Items** and **OperationalItems**; however, treating spare components, in this sense, also modifies the functional transform too. It introduces the functionality to acquire, manage, and use these spare components, which now introduces a distraction at best, or complexity in representing behavior. A logical equivocation occurs, unless the function definition explicitly changes. The "repair" function must conceptually become "repair, acquire, manage, and use spare components." The function becomes more complex and distracts from understanding and representing what the core functionality of the component is—it tends to make behavioral views harder to develop. To counter this tendency, MBSE offers the concept of a **Resource**—a resource in a different sense than used in the DoDAF literature. Here a **Resource** may be used to affect execution behavior to better understand the effects of resource limitations on the overall system performance.

[12] The usage of the term capability is as described in the DoD Architecture Framework, Version 2.02, 28 May 2009. In DoD-oriented models, capabilities refer to operationally oriented scenarios and threads refer to system-oriented scenarios.

Capabilities,[13] in general, are the starting point for defining operational scenarios. These scenarios consist of a sequence of **OperationalActivities** needed to satisfy the **Capability**. Each scenario of an **OperationalActivity** sequence begins with an external stimulus and each scenario ends with the provision of an external stimulus.[14] 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 **OperationalActivity** is *allocated to* 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, expressed as a sequence of **OperationalActivities** into a single behavior view 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 view 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 *allocated to* the context-level **Performer** (of Type Context) with the relationship attribute Behavior Type is set to "Integrated (Root)."[15]

OperationalActivity Inputs and Outputs. Each **OperationalActivity** within a behavioral view 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 *allocated to* relationship. This not only establishes the organization or role that performs the activity, it allows the systems engineering/architecture team to assess the impact of **Performer** losses or failures on both **Mission** and **OperationalActivities**, thereby, making it easier for the systems 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**.

Note: When developing behavior, a root **OperationalActivity** may be established for any **Performer** and the behavior diagram resulting from integrating the capability based **OperationalActivities** define the full behavior of the **Performer** from the **Performer's** perspective, which satisfies both the **Performer's** external observables and its allocated capabilities.

---

[13] There may be one or more *capability requirement* establishing the programmatic need and timeframe when the capability is needed. Capability Requirements are captured in the **Requirement** class of Type: **Capability**. In turn, this capability requirement specifies a capability in the **Capability** class.

[14] Sometimes the external stimulus upon output is not provided because the behavior is internally satisfied within the component.

[15] CORE's Model Assistant may automate the creation of the Integrated (Root) **OperationalActivity** upon creation of a **Performer** element.

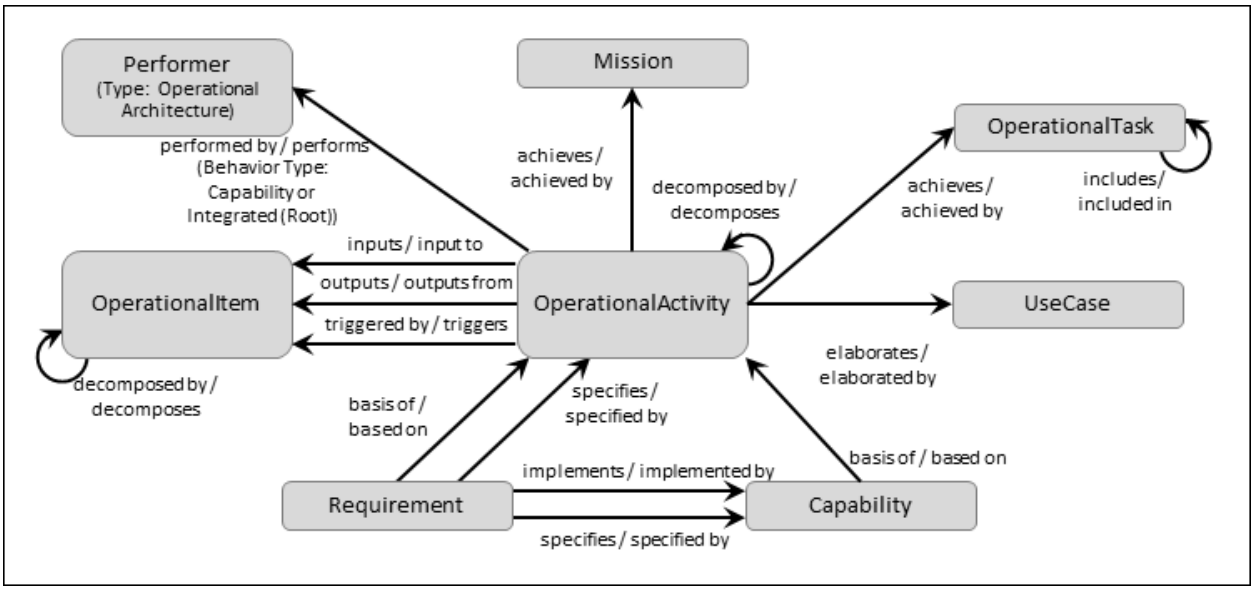

Figure 9 Operational Activity View

Table 6 Operational Activity View

Element Class	Attributes	Relationships	Target Classes
Capability	See Section 3.2	*based on / basis of*	**OperationalActivity**
Mission	See Section 3.2	*achieved by / achieves*	**OperationalActivity**
Performer (Type: Operational Architecture)	See Section 3.1	*performs / allocated to* (Behavior Type: Capability or Integrated (Root))	**OperationalActivity**
OperationalActivity	BeginLogic Description Doc. PUID Duration EndLogic ExitLogic Number Timeout Title	*achieves / achieved by*	**Mission** **OperationalTask**
		based on / basis of	**Capability** **OperationalActivity** **Requirement**
		basis of / based on	**OperationalActivity**
		decomposed by / decomposes	**OperationalActivity**
		elaborates /elaborated by	**UseCase**
		inputs / input to	**OperationalItem**
		outputs / output from	**OperationalItem**
		allocated to / performs (Behavior Type: Capability or Integrated [Root])[16]	**Performer**

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

Table 6 Operational Activity View

Element Class	Attributes	Relationships	Target Classes
		results in/ result of	**Capability** **Requirement**
		specified by / specifies	**Requirement**
		triggered by / triggers	**OperationalItem**
OperationalItem	Accuracy Description Doc. PUID Number Priority Timeliness Type	*decomposed by / decomposes*	**OperationalItem**
		implemented by / implements	**Item**
		input to / inputs	**OperationalActivity**
		output from / outputs	**OperationalActivity**
		specified by / specifies	**Requirement**
		transferred by / transfers	**Needline**
		triggers / triggered by	**OperationalActivity**
OperationalTask	See Section 3.2	*achieved by / achieves*	**OperationalActivity**
		achieves / achieved by	**Mission**
Requirement	See Section 3.2	*basis of / based on*	**OperationalActivity**
		specifies /specified by	**Architecture** **Capability** **OperationalActivity** **OperationalItem** **Performer**
UseCase	Alternate Flow Description Number Preconditions Primary Flow Postconditions	*augmented by / augments*	**ExternalFile**
		describes / described by	**Performer**
		elaborated by / elaborates	**OperationalActivity**
		elicits / elicited by	**Requirement**
		extended by / extends	**UseCase**
		generalization of / kind of	**UseCase**
		included in / includes	**UseCase**
		involves / participates in	**Performer**

4.2 State View

A **State** viewpoint offers an alternative approach for expressing a component's behavior, the identification of relative functional timing of a **Component** or state machine. A **State** identifies a non-overlapping (i.e., one state does not share its behavior with another state) operational and possibly repetitive conditions occurring during component's operating lifetime. In other words, the set of **States** *exhibited by* a **Component** are complete for expressing a **Component's** behavior and its logical timing (not absolute

timing). Alternative **State** representations are possible, but each set definition must be complete and non-overlapping. Associated with the *exhibits / exhibited by* relation pair is a behaviorType attribute; the values of which are: Atomic and Integrated (Root).

A **State** may exist either because it is *documented by* a **Document** or *specified by* a **Requirement**. An **ExternalFile** or **Text** element may also *augment* a **State** for the purpose of further enhancing the meaning or representation of the **State**.

A given **State** may be a member of a particular subset of **States**. The collection of such **States** is represented as a **Mode**; consequently a **State** is *encompassed by* a **Mode**. Since a **Performer** *exhibits* a **State**, therefore, a **Performer** also *contains* a **Mode**. Each **State** *incorporates* one or more **OperationalActivities**. Associated with the *incorporates* relation is a behaviorType attribute. A value of "Atomic" infers that the target **OperationalActivity** will be non-interruptible from an execution perspective.

A value of "Integrated (Root)" infers that a functional behavioral model applies to the behavior within the **State** machine diagram. The remaining two values for the behaviorType attribute, Entry and Exit, refer to which **State** is the initial or final **State** respectively on the **State** machine diagram.

One or more subordinate **States** may *decompose* a single **State**, which delineates the progression from a composite **State** to an atomic **State** (the targets of the *decomposed by* relationship is empty). The movement from one **State** to another **State** occurs through a **Transition**. A **State** is *exited by* a **Transition** and correspondingly, the **Transition** enters a new **State** or may re-enter the same **State**. However, the timing of the Transition's effect is governed by a Guard condition attribute. The Guard condition attribute is a rule, empty, simple, or complex, which results in a Boolean value (an empty Guard condition is not evaluated). If true, the transition occurs; otherwise, the transition waits for the Guard condition to change from false to true.

Events serve to communicate to External **State** machines at the time point of a **Transition**. A **Transition** *triggers* an **Event** and an **Event** is *responsible for* an **OperationalItem**, which conveys the message governed by the **Event**.

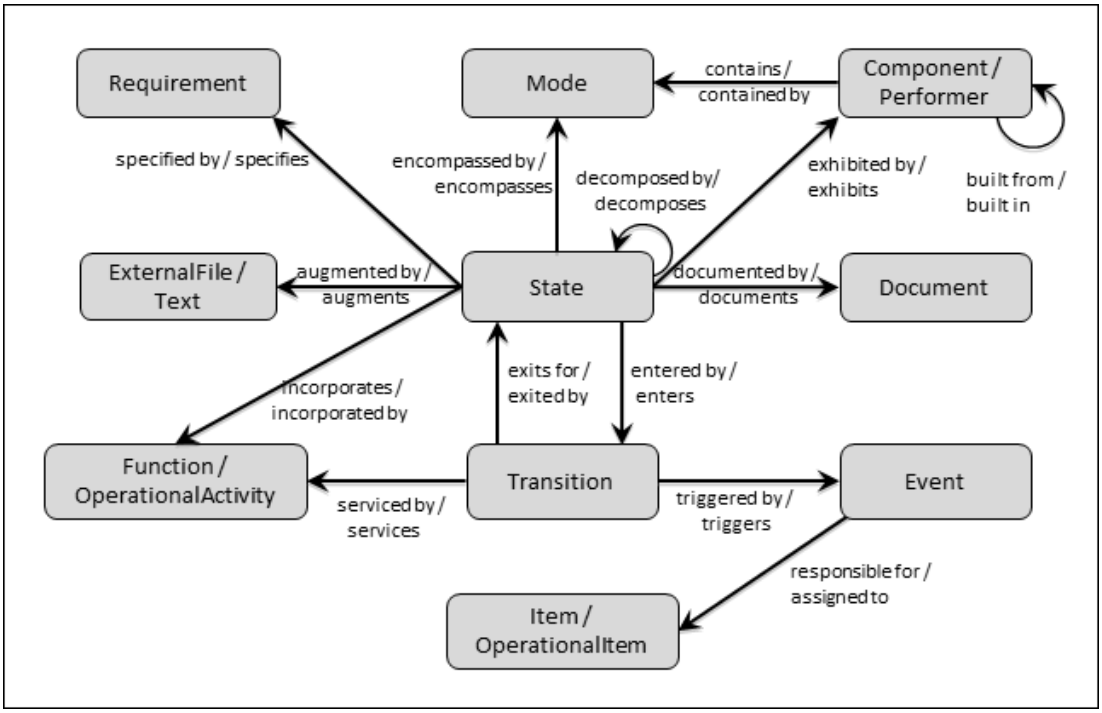

Figure 10 State View

Table 7 State View

Element Class	Attributes	Relationships	Target Classes
Component	See SDG	*contains / contained by*	**Mode**
		exhibits / exhibited by	**State**
Event	Description Doc. PUID	*augmented by / augments*	**ExternalFile** **Text**
		documented / documents	**Document**
		responsible for / assigned to	**Item** **OperationalItem**
		triggers / triggers	**Transition**
Function	See SDG	*incorporated by / incorporates*	**State**
		services / serviced by	**Transition**
Item	See SDG	*assigned to / responsible for*	**Event**
Mode	Description Number	*augmented by / augments*	**ExternalFile** **Text**
		contained by / contains	**Component** **Performer**
		documented by / documents	**Document**
		encompasses / encompassed by	**State**
		impacted by / impacts	**Concern** **Risk**
		specified by / specifies	**Requirement**
OperationalActivity	See Section 4.1	*incorporated by / incorporates*	**State**
		services / serviced by	**Transition**
OperationalItem	See Section 4.1	*assigned to / responsible for*	**Event**
Performer	See Section 3.1	*contains / contained by*	**Mode**
State	Description Doc. PUID Number Title	*augmented by / augments*	**ExternalFile** **Text**
		decomposed by / decomposes	**State**
		documented by / documents	**Document**
		encompassed by / encompasses	**Mode**
		entered by / enters	**Transition**

Element Class	Attributes	Relationships	Target Classes
		exhibited by / exhibits	**Component** **Performer**
		exits by / exits for	**Transition**
		impacted by / impacts	**Concern** **Risk**
		incorporates / incorporated by	**Function** **OperationalActivity**
		specified by / specifies	**Capability** **Requirement**
Transition	Delay Delay Units Description Guard Number	*augmented by / augments*	**ExternalFile** **Text**
		documented by / documents	**Document**
		enters / entered by	**State**
		exit for / exits by	**State**
		triggered by / triggers	**Event**

5. OPERATIONAL ARCHITECTURE SYNTHESIS

5.1 Assign OperationalActivities to Next Level of Performers

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

As the **Performer** hierarchy evolves, **Performers** uniquely *perform* more specific **OperationalActivities**. **OperationalActivity** refinement is accomplished in layers. When a decomposed root or capability **OperationalActivity** is *allocated to* a **Performer**, all lower-level **OperationalActivities** in its decomposition path are part of the behavior of the **Performer**. The **Performer** may be correspondingly decomposed, in which case even lower-level **OperationalActivities** are allocated to the lower-level **Performers**. The lowest-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 4.1 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 4.1, when developing behavior, a root **OperationalActivity** can be established for any **Performer** and the behavior diagram constructed using the atomic **OperationalActivities**, which defines the full behavior of the **Performer** from the **Performer's** perspective rather than from the architecture's perspective.

[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.

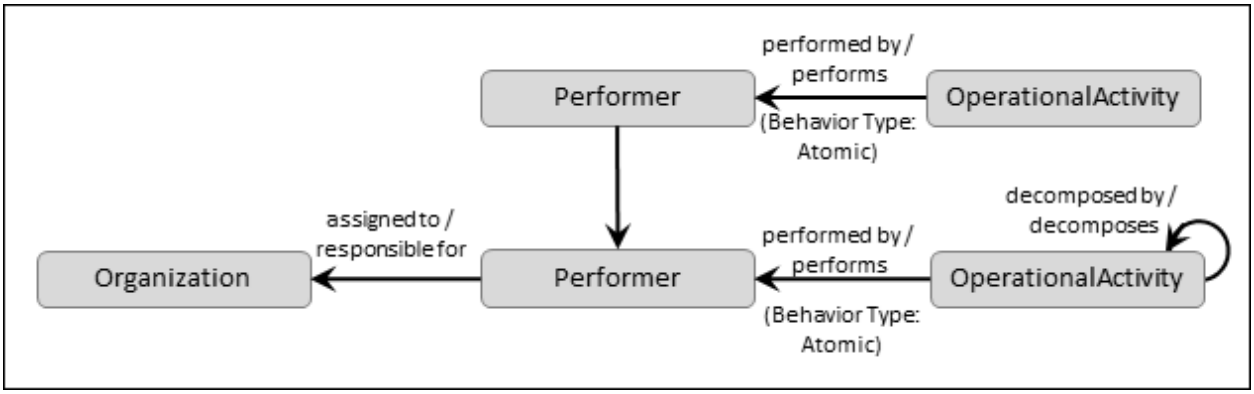

Figure 11 Performer Hierarchy and OperationalActivity Assignment

Table 8 Performer Hierarchy and OperationalActivity Assignment

Element Class	Attributes	Relationships	Target Classes
Performer	See Section 3.1	*assigned to / responsible for*	**Organization**
		built from / built in	**Performer**
		built in / built from	**Performer**
		performs / allocated to	**OperationalActivity**
OperationalActivity	See Section 4.1	*allocated to / performs*	**Performer**
Organization	See Section 3.3	*responsible for / assigned to*	**Performer**

5.2 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 3.4).[18] **Needlines** are decomposable by means of the *includes* relationship. Since a **Needline** has a maximum of two targets, a decomposable **Needline** enables the systems engineer/architect to make the **Needline** connections consistent with the **Performer** hierarchy, without having to move a terminus point of a **Needline** to a lower-level **Performer**. This simplifies the maintenance of **Needlines** through the architecture.

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

[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.

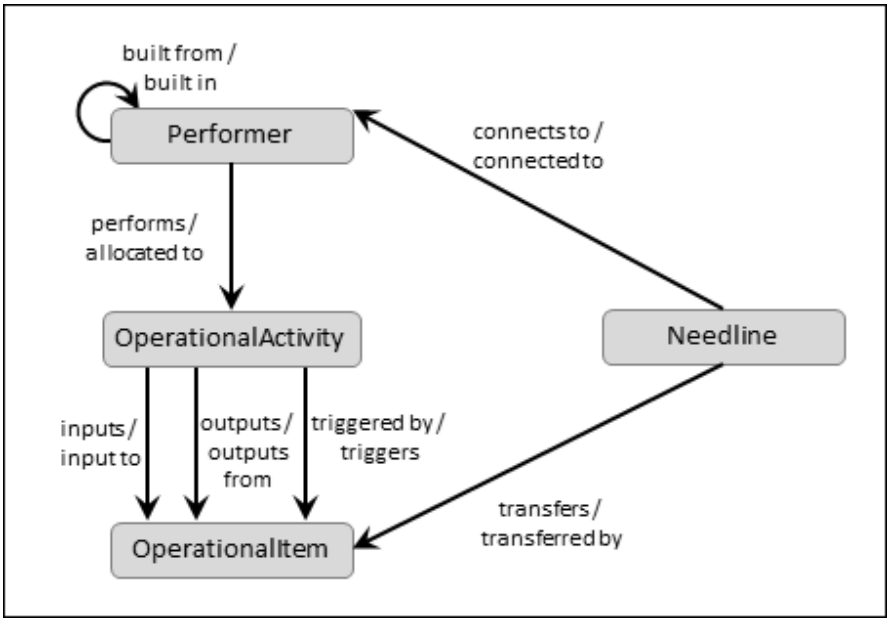

Figure 12 External Needline Definition

Table 9 External Needline Definition

Element Class	Attributes	Relationships	Target Classes
Needline	See Section 3.4	*connects to / connected to*	**Performer**
		includes / included in	**Needline**
		transfers / transferred by	**OperationalItem**
OperationalItem	See Section 4.1	*transferred by / transfers*	**Needline**
		input to / inputs	**OperationalActivity**
		outputs / output from	**OperationalActivity**
		triggers / triggered by	**OperationalActivity**
Performer	See Section 3.1	*connected to / connects to*	**Needline**
		built from / built in	**Performer**

5.3 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 repository using the **Needline** element class.

Needlines are decomposable by means of the *includes* relationship. Since a **Needline** has a maximum of two targets, a decomposable **Needline** enables the systems engineer/architect to make the **Needline** connections consistent with the **Performer** hierarchy, without having to move a terminus point of a **Needline** to a lower-level **Performer**. This simplifies the maintenance of **Needlines** through the architecture.

As the **Performer** hierarchy evolves further, the lower-level **Performers,** terminating a **Needline**, *perform* **OperationalActivities** and these *outputs*, *inputs*, or *triggered by* the **OperationalItems** transferred by the **Needlines**.

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

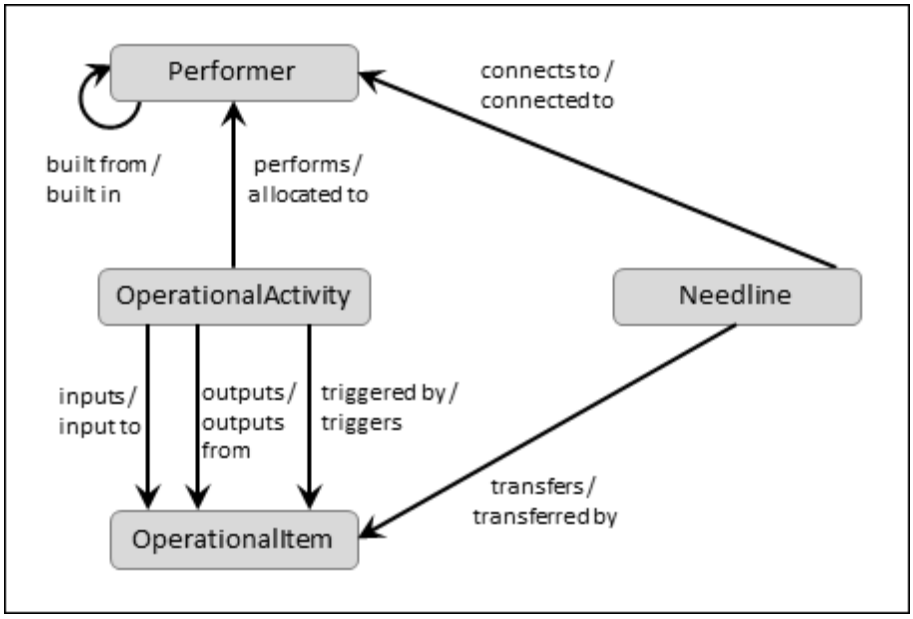

Figure 13 Internal Needline Definitions

Table 10 Internal Needline Definitions

Element Class	Attributes	Relationships	Target Classes
Needline	See Section 3.4	*connects to / connected to*	**Performer**
		includes / included in	**Needline**
		transfers / transferred by	**OperationalItem**
OperationalItem	See Section 4.1	*transferred by / transfers*	**Needline**
		input to / inputs	**OperationalActivity**

Table 10 Internal Needline Definitions

Element Class	Attributes	Relationships	Target Classes
		output from / outputs	**OperationalActivity**
		triggers / triggered by	**OperationalActivity**
Performer	See Section 3.1	*connected to / connects to*	**Needline**
		performs / allocated to	**OperationalActivity**

6. OPERATIONAL VIEWPOINT VALIDATION USING CORESIM

CORESim is a discrete event simulator that executes the **OperationalActivity** and **Needline** behavior views to provide an assessment of operational architecture performance and to verify the dynamic integrity of the conceptual model. CORESim dynamically interprets a behavior view (i.e., an Activity Diagram, Enhanced Functional Flow Block Diagram [EFFBD]) in conjunction with **Needlines** and identifies and displays timing, resource utilization, operational item flow, and behavioral inconsistencies. CORESim usage should be an integral part of operational analysis and operational architecture synthesis.

7. OPERATIONAL ARCHITECTURE CONSIDERATIONS

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

7.1 Performance Requirements

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

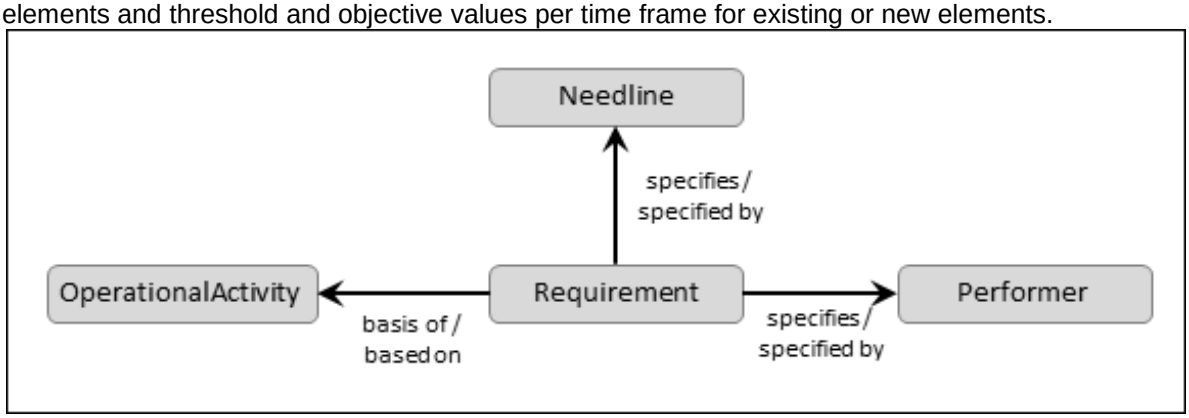

Figure 14 Performance Requirements

Table 11 Performance Requirements

Element Class	Attributes	Relationships	Target Classes
Requirement	See Section 3.2	*basis of / based on*	**OperationalActivity**
		specifies / specified by	**Performer** **Needline**

7.2 Services Development

Services exist as both a subset of functional behavior and as part of a system. Within the functional behavior view [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 *documents* a **ServiceSpecification**.

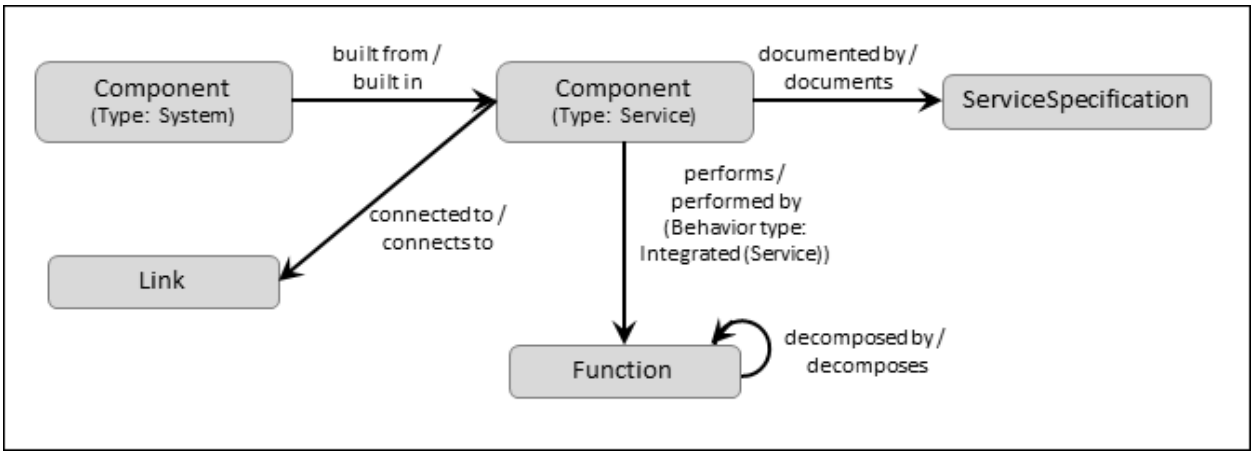

Figure 15 Services

Table 12 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 / allocated to (Behavior Type: Integrated (Services)	**Function**
		documented by / documents	**ServiceSpecification**
Function	See SDG	allocated to / performs (Behavior Type: Integrated (Services)	**Component**
Link	See SDG	*connected to / connects to*	**Component**

Table 12 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)

7.3 Requirements Development

Capabilities, OperationalActivities and **UseCases** serve as sources for system **Requirements**. **Capabilities**, more typically are associated with **OperationalActivities,** which lead to the identification and definition of functional **Requirements**. However, **Capabilities**, in themselves, may directly lead to system **Requirements**. **UseCases** lead to the identification and definition of functional and 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 **UseCase**. See the SDG for a description and use of **Requirement** attributes.

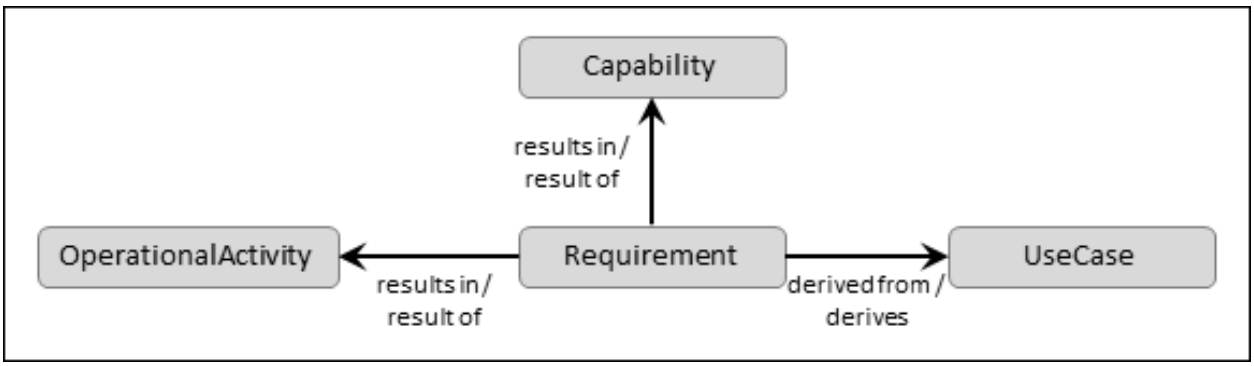

Figure 16 Requirements Development

Table 13 Requirements Development

Element Class	Attributes	Relationships	Target Classes
OperationalActivity	See Section 4.1	*results in / result of*	**Requirement**
Requirement	See Section 3.2	*result of / results in*	**OperationalActivity**

7.4 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 systems 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 systems 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**.

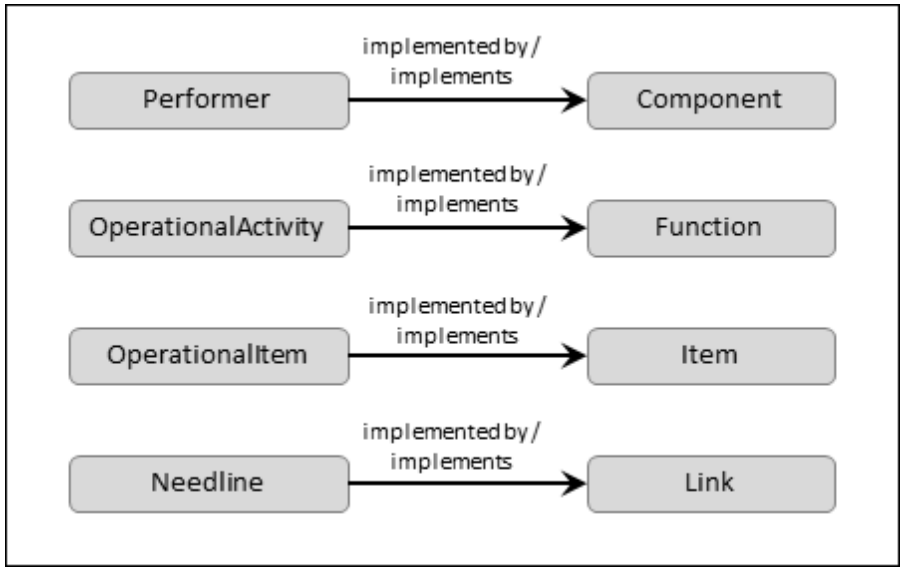

Figure 17 Operational to Systems Traceability

Table 14 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**

Table 14 Operational to Systems Traceability

Element Class	Attributes	Relationships	Target Classes
Needline	See Section 3.4	*implemented by / implements*	**Link**
OperationalActivity	See Section 4.1	*implemented by / implements* (Status: nil, Planned, Partial, or Full)	**Function**
OperationalItem	See Section 4.1	*implemented by / implements*	**Item**
Performer	See Section 3.1	*implemented by / implements*	**Component**

8. 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-v2.02-based enterprise architectural descriptions.

8.1 Program/Project Basics

Organizations and **Architectures** are related through the Program/Project Viewpoint to relate the enterprise's objectives with the **Architecture** and those **Organizations** involved. The Program or Project viewpoint 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**.[19] *Assigned to* each **ProgramElement** is an **Organization,** which is responsible for some aspect of the program/project. A **ProgramElement** of Type: Task represents the lowest **ProgramElement** for which cost accounting is performed.

The top-most **ProgramElement** is *specified by* one or more programmatic **Requirements**, which are represented as elements within the **Requirement** class. These **Requirements** describe the desired effect (outcome) or achievement level in operational processes, projects, or special programs. Subordinate **Requirements** may *specify* lower-level **ProgramElements** (Type: Project, Work Package or Task). 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 4.1). A **ProgramElement** also supplies one or more stakeholder

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

deliverables (i.e., an element of some or all of these classes **Capability**, **Component**, **Document** or **Performer**).

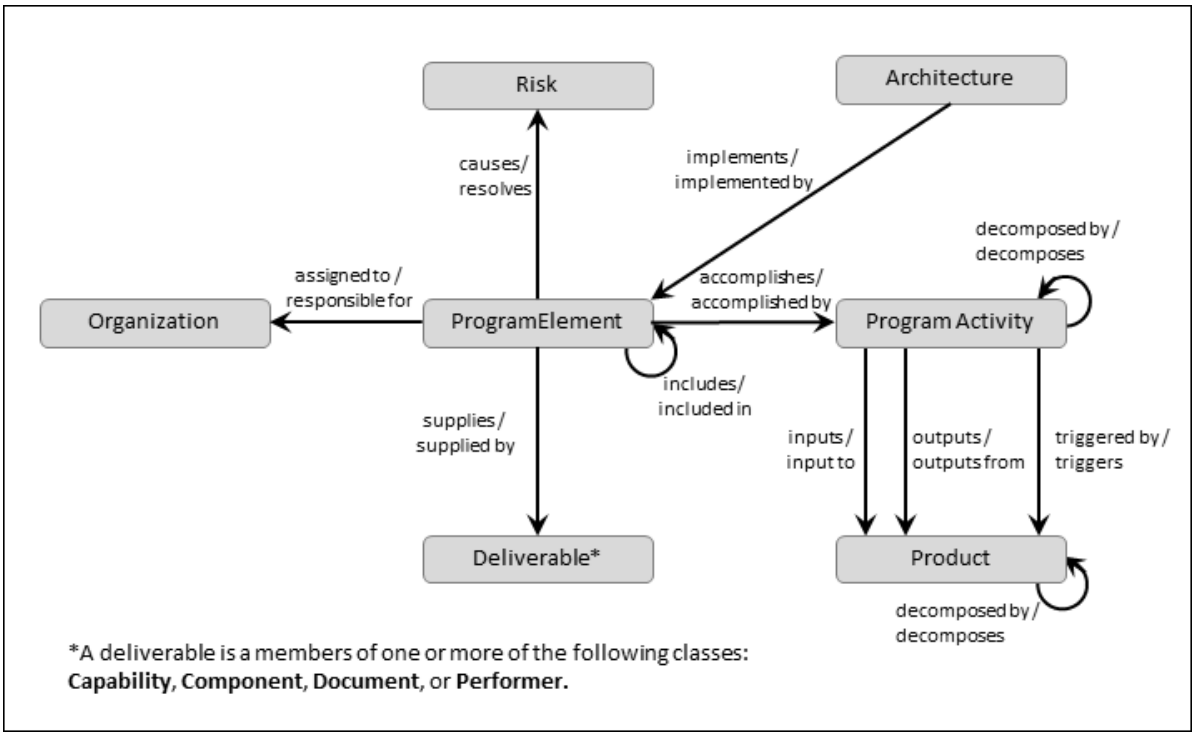

Figure 18 Program Management Basics

Table 15 Program Management Basics

Element Class	Attributes	Relationships	Target Classes
Architecture	See Section 3.1	*implemented by / implements* (Status: nil, Planned, Partial, or Full)	**ProgramElement**
Capability	See Section 3.2	*supplied by / supplies*	**ProgramElement**
Component	See SDG	*supplied by / supplies*	**ProgramElement**
Document	See Section 3.2	*supplied by / supplies*	**ProgramElement**
Performer	See Section 3.1	*supplied by / supplies*	**ProgramElement**
Organization	See Section 3.3	*responsible for / assigned to*	**ProgramElement**
ProgramElement	Contract Number Cost Description End Date	*accomplishes / accomplished by*	**ProgramActivity**
		assigned to / responsible for	**Organization**

Table 15 Program Management Basics

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

8.2 Program Management Activity View

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 view that fully represents the workflow required by the parent **ProgramActivity**. COREsim (see Section 6) will execute the program activity views to provide an assessment of the timeline performance (schedule) and to verify the dynamic integrity of the conceptual program management view. COREsim dynamically interprets a behavior view (i.e., the EFFBD) and identifies and displays timing, resource usage, product flow, and viewpoint 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.

Requirements Traceability. **ProgramActivity** traceability to an appropriate **Requirement** element is established using the *based on* relationship. Traceability to a **ProgramElement** uses the *accomplishes* relationship thereby, identifying the **ProgramActivities** that accomplish some aspect of the work breakdown structure.

ProgramActivity traceability from an appropriate **Capability** occurs through the *supplied by* relationships to an intermediary **ProgramElement**. The **ProgramElement** is *accomplished by* one or more **ProgramActivities** that apply for achieving that **Capability**.

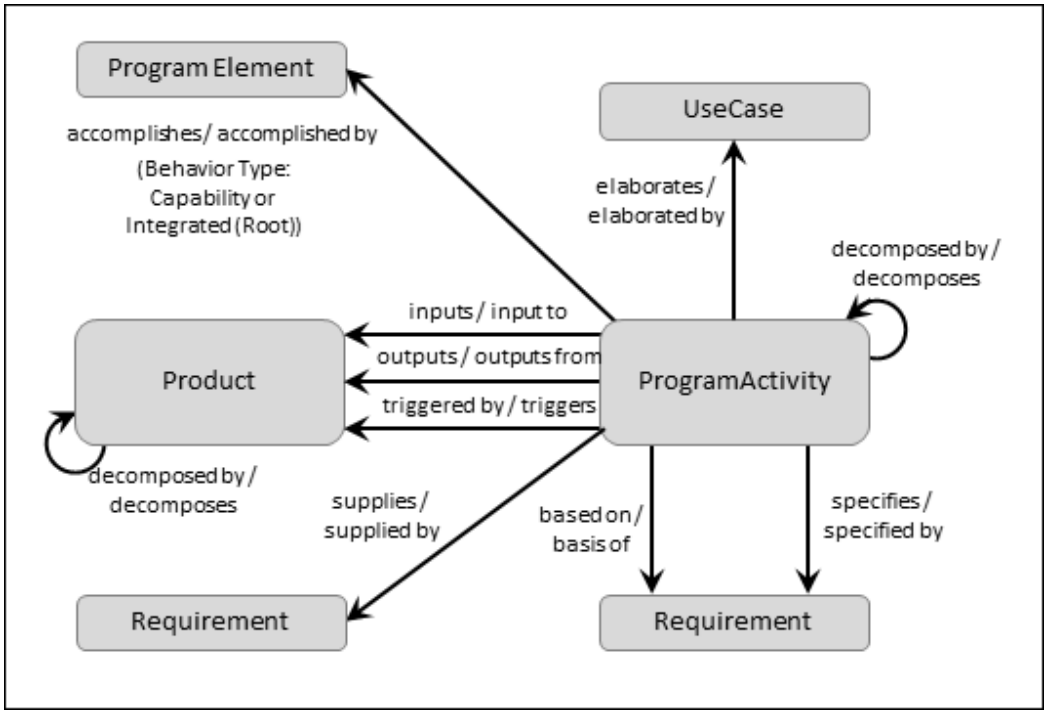

Figure 19 Program Activity View

Table 16 Program Activity View

Element Class	Attributes	Relationships	Target Classes
Capability	See Section 3.2	*supplied by / supplies*	**ProgramElement**
Component	See SDG	*supplied by / supplies*	**ProgramElement**
Performer	See Section 3.1	*supplied by / supplies*	**ProgramElement**
ProgramElement	See Section 8.1	*accomplishes / accomplished by*	**ProgramActivity**
		assigned to / responsible for	**Organization**
		augmented by / augments	**ExternalFile**
		causes / resolves	**Risk**
		implements / implemented by	**Architecture**
		included in / includes	**ProgramElement**
		resolves / causes	**Risk**
		specified by / specifies	**Requirement**

Table 16 Program Activity View

Element Class	Attributes	Relationships	Target Classes
		supplies / supplied by	**Capability** **Component** **Document** **Performer**
Product	Description Number Size Size Units Type	*augmented by / augments*	**ExternalFile**
		decomposed by / decomposes	**Product**
		documented by / documents	**Document**
		input to / inputs	**ProgramActivity**
		output from / outputs	**ProgramActivity**
		specified by / specifies	**Requirement**
		triggers / triggered by	**ProgramActivity**

9. DOCUMENTATION—DODAF V2.02 VIEWPOINTS

CORE includes a set of scripts to output each of the DoDAF v2.02 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 MBSE process to architecture definition, these scripts have been designed to be flexible in order to support the architects/systems engineers developing the architecture on an on-going basis and to produce the viewpoints for customer usage.

Table 17 DoDAF v2.02 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 supplies Capability, and **Capability** is *refined by* **Capability.**
CV-3	Capability Phasing	User selected **Architecture** *implemented by* **ProgramElement**, which *supplies* **Capabilities**. **ProgramElements** determine when projects supplying elements of capability are to be delivered, upgraded and/or withdrawn.
CV-4	Capability Dependencies	**Category** *categorizes* **Capability**
CV-5	Capability to Organizational	User selected **Architecture specified by Capability**

Table 17 DoDAF v2.02 Viewpoint Scripts

Viewpoint	Viewpoint Title	Script Output
	Development Mapping	**refined by Capability**
CV-6	Capability to Operational Activities Mapping	User selected **Architecture** *specified by* **Capability** *refined by* **Capability** *basis of* **OperationalActivity** *allocated to* **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 stemming from the **Architecture** *composed of* **Component** of Type, perform **Functions** of Type: Service.
DIV-2	Logical Data Model	Outputs characteristics of **OperationalItems** that are *output from*, *input to*, or *triggers* one or more **OperationalActivities**, which are derived from a user selected **Architecture** *composed of* **Performers**. A **Performer** *performs* an **OperationalActivity**, its children, and, optionally, their children.
DIV-3	Physical Data Model	Outputs an **OperationalItem** characteristics table for **OperationalItems** related to a user selected **Architecture**, **OperationalItems** are derived from a user selected **Architecture** *composed of* **Performers**. A **Performer** performs an **OperationalActivities**, its children, and, optionally, their children.
OV-1	High-Level Operational Concept Graphic	Outputs a hierarchy diagram and/or **ExternalFile** for each **Performer** composing an **Architecture**.
OV-2	Operational Resource Flow Description	Physical Block Diagram (PBD) for each **Performer** composing an **Architecture.**
OV-3	Operational Resource Flow Matrix	Summary matrix or full matrix for information exchanges of the children of **OperationalActivity**(s) *allocated to* **Performers** composing the user selected **Architecture**.
OV-4	Organization Relationships Chart	Organization Hierarchy stemming for each **Architecture** *assigned to* an **Organization**.
OV-5a	Operational Activity Decomposition Tree	Capability to Operational Activities Hierarchy for **Capability** *specifies* **Architecture** Performer Operational Activities Hierarchy for **OperationalActivity**(s) *allocated to* **Performers** that *compose* the user selected **Architecture**.
OV-5b	Operational Activity Model	User selected behavior diagram for presenting an **OperationalActivity** and its children. Includes optional output of Function Hierarchy for selected **OperationalActivity**.
OV-6a	Operational Rules Model	EFFBD or Activity Diagrams for **OperationalActivity(s)** *allocated to* **Performers** that *compose* the user selected **Architecture**, with **Requirements** of Type: Guidance, which specifies one or more **OperationalActivities**.
OV-6b	State Transition Description	User selected **ExternalFiles** and **States** that are *exhibited by* **Performers** that *compose* the user selected **Architecture**.

Table 17 DoDAF v2.02 Viewpoint Scripts

Viewpoint	Viewpoint Title	Script Output
OV-6c	Event-Trace Description	Sequence Diagrams for **OperationalActivity(s)** *allocated to* **Performers** that *compose* the user selected **Architecture**.
PV-1	Project Portfolio Relationships	**Organization** linked to **ProgramElement** and one hierarchical level below the top **ProgramElement** to account for Projects subordinate to a Program.
PV-2	Project Timelines	An **ExternalFile** and **ProgramElement** Table derived from a user selected **Architecture**.
PV-3	Project to Capability Mapping	User selected **Architecture** implemented by **ProgramElements** mapped to **Capabilities**.
SvcV-1	Services Context Description	Hierarchy and Interface Block Diagrams for **Component(s)** of type Service that *composes* the user selected **Architecture.** Also, an Element Definition and Interconnection Table for the **Components**, **Interfaces**, **Items** and **Links** encountered.
SvcV-2	Services Resource Flow Description	Physical Block Diagram and Resource Flow Table for **Component(s)** of type Service that *composes* the user selected **Architecture**.
SvcV-3a	Systems-Services Matrix	Matrix identifying interfacing **Component(s)** of type Service with those **Component(s)** that are not of type Service that *composes* the user selected **Architecture**.
SvcV-3b	Services-Services Matrix	Matrix identifying interfacing **Component(s)** of type Service that *composes* the user selected **Architecture.**
SvcV-4	Services Functionality Description	User selected behavior diagrams and tables for **Function(s)** *allocated to* **Component(s)** of type Service that *composes* the user selected **Architecture.**
SvcV-5	Operational Activity to Services Traceability Matrix	Matrix mapping **Functions** *allocated to* **Component(s)** type Service that *composes* the user selected **Architecture** and their associated **Links**, **Functions,** and **Interfaces** to **OperationalActivity(s).**
SvcV-6	Services Resource Flow Matrix	Summary matrix or full matrix for data exchanges of the children of **Component (s)** of type Service that *composes* the user selected **Architecture**.
SvcV-7	Services Measures Matrix	Quantitative performance measures (**Requirements** of Type performance) for the children of **Component(s)** of type Service that *composes* the user selected **Architecture** and their associated **Interfaces, Links,** and **Functions**.
SvcV-8	Services Evolution Description	User selected **ExternalFile**.
SvcV-9	Services Technology & Skills Forecast	User selected **ExternalFile**.
SvcV-10a	Services Rules Model	Table for **Document(s)** of type Guidance or Standard, which *document* **Requirement(s)**. A second table, which *document* **Components**, **Functions**, **Interfaces**, **Items** and **Links**.

Table 17 DoDAF v2.02 Viewpoint Scripts

Viewpoint	Viewpoint Title	Script Output
SvcV-10b	Services State Transition Description	Hierarchy Diagram of **Components** of type Service that *exhibits* **State(s)** that *composes* the user selected **Architecture**.
SvcV-10c	Services Event-Trace Description	Sequence Diagram and table for service related **Functions** *allocated to* **Component(s)** of type Service that *composes* the user selected **Architecture**.
StdV-1	Standards Profile	A table 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 and Interconnection Table for **Component(s)** type System that *composes* user selected **Architecture**.
SV-2	Systems Resource Flow Description	Physical Block Diagram and Resource Flow Table for **Component(s)** of type System that *composes* user selected **Architecture**.
SV-3	Systems-Systems Matrix	Matrix identifying interfacing **Component(s)** of types other than Service that *composes* the user selected **Architecture**.
SV-4	Systems Functionality Description	User selected behavior diagrams and tables for **Function(s)** *allocated to* **Component(s)** type System that *composes* the user selected **Architecture**.
SV-5a	Operational Activity to Systems Function Traceability Matrix	Matrix mapping Functions *allocated to* **Component(s)** of type System that *composes* the user selected **Architecture** and their associated **OperationalActivity(s), Performers** and **Capabilities.**
SV-5b	Operational Activity to Systems Traceability Matrix	Matrix mapping **Component(s)** of 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)** of type System that *composes* the user selected **Architecture**.
SV-7	Systems Measures Matrix	Quantitative performance measures (**Requirements** of Type performance) for the children of **Component(s)** of type System that *composes* the user selected **Architecture** and their associated **Interfaces, Links,** and **Functions**.
SV-8	Systems Evolution Description	User selected **Component** and **ExternalFile**.
SV-9	Systems Technology & Skills Forecast	User selected **Component** and **ExternalFile**.
SV-10a	Systems Rules Model	EFFBD or Activity diagrams for **Function(s)** *allocated to* **Components(s)** of type System that *composes* the user selected **Architecture**.
SV-10b	Systems State Transition	Hierarchy Diagram of **Components** of type System that

Table 17 DoDAF v2.02 Viewpoint Scripts

Viewpoint	Viewpoint Title	Script Output
	Description	*exhibits* **State(s)** that *composes* the user selected **Architecture**.
SV-10c	Systems Event-Trace Description	Sequence Diagram and table for system related **Functions** *allocated to* **Component(s)** of 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, etc. These should be used on an on-going basis to aid the systems engineers in communication and assessment of the architecture definition.

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