Architecture Definition Guide

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 2017

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CUSTOMER RESOURCE OPTIONS

Supporting users throughout their entire journey of learning model-based systems engineering (MBSE) is central to Vitech's mission. For users looking for additional resources outside of this document, please refer to the links below. Alternatively, all links may be found at www.vitechcorp.com/resources.

Webinars

Webinar archive with over 40 hours of premium industry and tool-specific content.

Screencasts

Short videos to guide users through installation and usage of Vitech software.

A Primer for Model-Based Systems Engineering

Our free eBook and our most popular resource for new and experienced practitioners alike.

Help Files

Searchable online access to Vitech software help files.

Technical Papers

Library of technical and white papers for download, authored by Vitech systems engineers.

MySupport

Knowledge Base, Exclusive Webinars and Screencasts, Chat Support, Documents, Download Archive, etc.

Our team has also created resources libraries customized for your experience level:

All Resources	**Advanced**
Beginner	**IT / Sys Admin**
Intermediate	**Student**

PREFACE

This Architecture Definition Guide (ADG) provides a structured approach for populating a GENESYS™ software project with operational architectural information and producing information about the architecture following the Department of Defense Architecture Framework (DoDAF) requirements for viewpoint generation using the reports provided with GENESYS. 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 GENESYS System Definition Guide (SDG).

An operational architecture contains operational entities, system entities, and program management entities, all of which must be considered in operational architecture development.[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 GENESYS 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, GENESYS can produce documentation for the purpose of presentation, review, and analysis of the architecture as well as integration and comparison with 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 GENESYS 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 using the reports provided with GENESYS.

This guide augments the SDG and the MBSE with GENESYS 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 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.

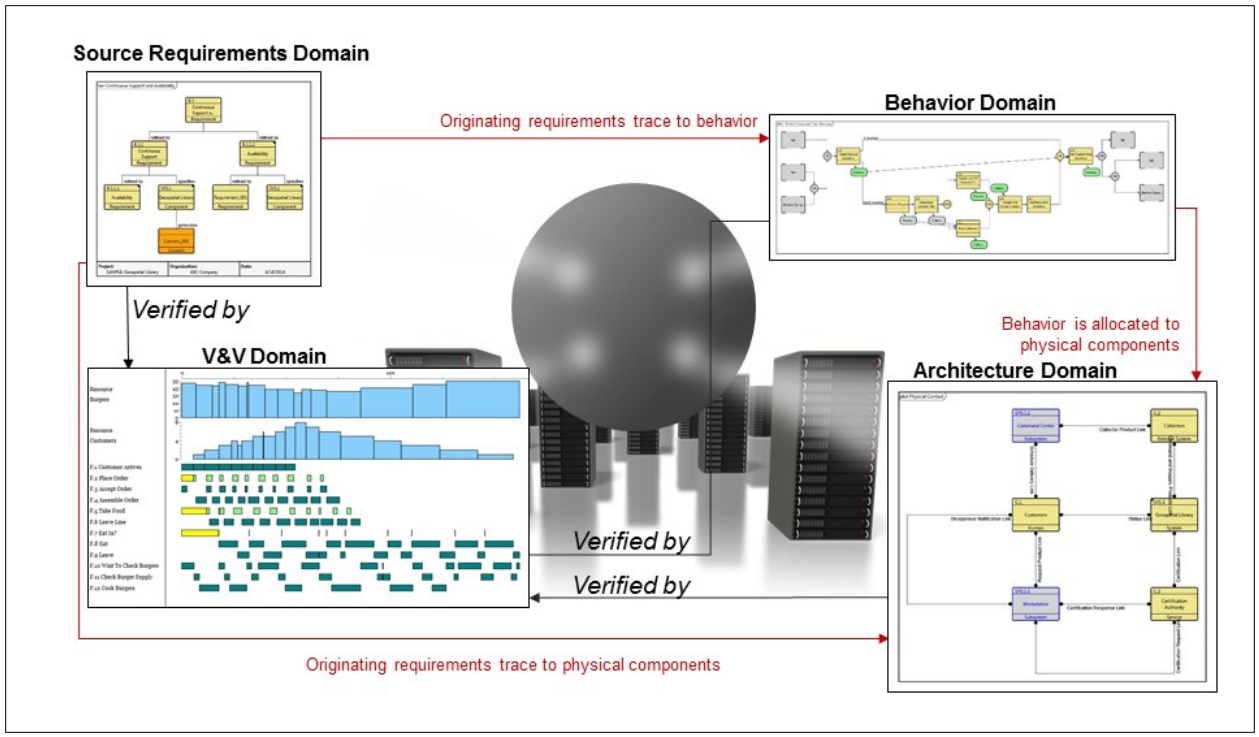

Figure 1 MBSE Activities

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

- For details on generating DoDAF v2.02 viewpoints, the reader is referred to the DoDAF Viewpoints Definition help document provided in the GENESYS Documentation folder.

THIS PAGE INTENTIONALLY BLANK

1. ARCHITECTURE CONCEPTS

As portrayed in Figure 2, the DoDAF v2.0 schema is organized to provide both 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.

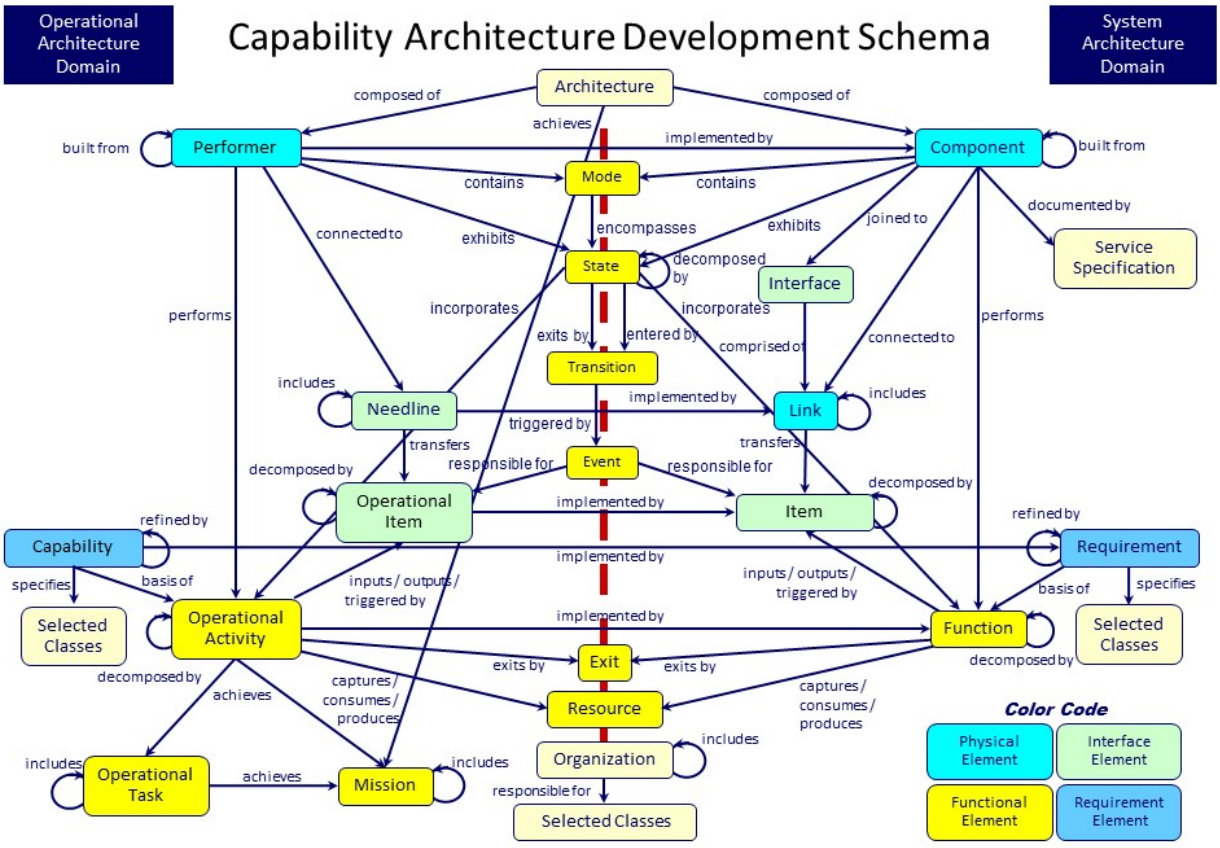

Figure 2 Capability Architecture Development Schema – Key Classes and Relationships of the Schema

As portrayed in Figure 3, the schema is extended to provide integration with the Program Management Domain. The Program Management Domain addresses the programmatic aspects of the operational/system architectures to assist in managing project and program efforts as well as finding commonality, duplicative, and missing capabilities among other architectures managed by a project office. These aspects help an executive/manager address duplication, misappropriation of scarce resources, and the timeliness of the delivered capabilities to the business enterprise.

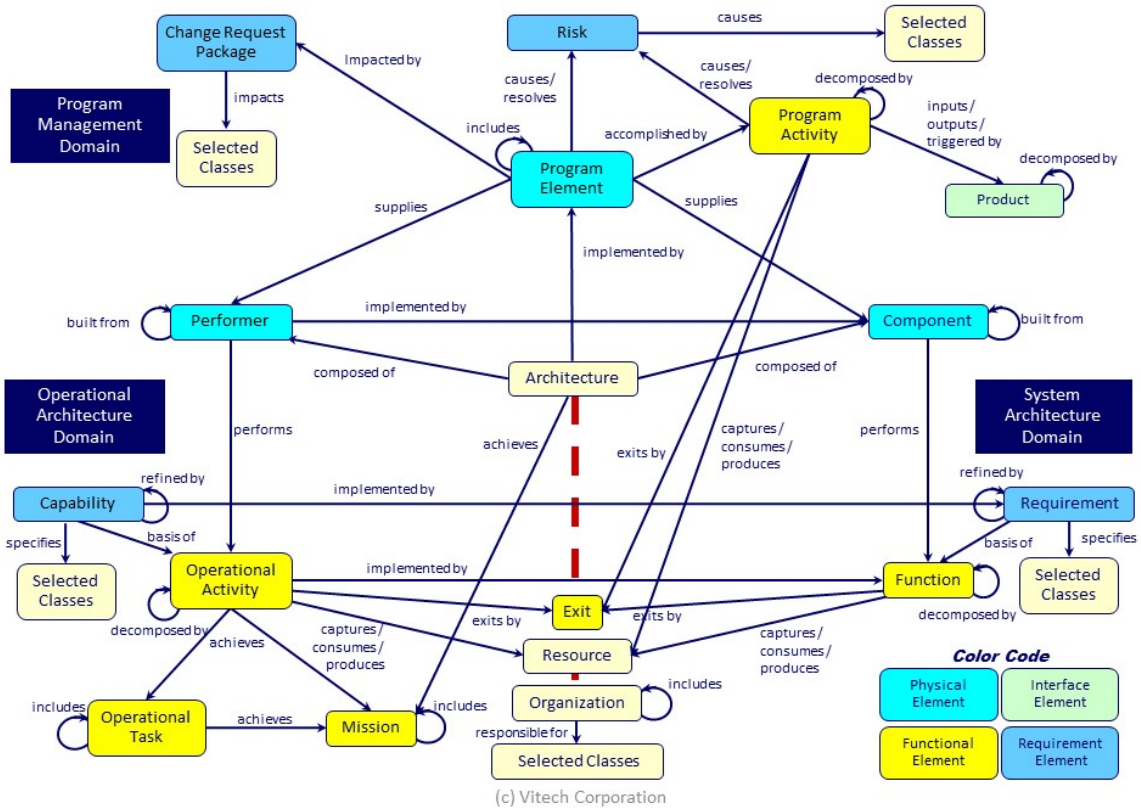

Figure 3 Capability Architecture Development Schema - Relationship between Operational, System, and Program Management Domains.

This ADG provides guidance into structuring the entities, attributes, and relationships that implement the Operational Architecture Domain and Program Management Domain for a project. Similarly, the SDG provides guidance into structuring the entities, attributes, and relationships that implement the System Architecture Domain.

1.1 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 entity which spans the two domains is specified in Section 2.1 *Define Architecture* and is composed of **Performer** (of type: Operational Architecture) and **Component** (of type: Family of Systems, System Architecture, or System of Systems) entities.

Within the Operational Architecture Domain, the **Performer** (type: Operational Architecture) is part of the operational context which also includes the **Performer** entities that represent the external aspects of the operational domain. See Section 2.4 *Define Operational Boundary* for details on defining the operational boundary.

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

2. 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 systems/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 high-level requirements that are treated as originating operational requirements and/or architectural guidance information (See Section 2.2 *Capture Source Material*).

2.1 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 in the operational architecture are represented in GENESYS using the **Performer** class. Physical entities, including collections of systems, interfacing systems, and entities within the systems architecture, are represented in GENESYS using the **Component** class. A **Performer** or **Component** Type attribute designates what the entity 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 entity or its relative position within the performer hierarchy.

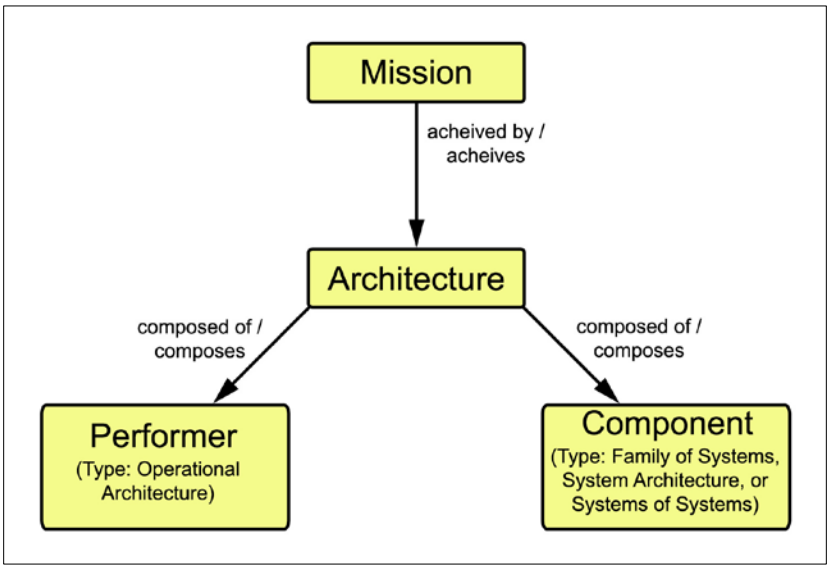

Figure 4 Architecture Definition[2]

Table 1 Architecture Definition

Entity Class	Attributes	Relations	Target Classes
Architecture	Description Number Purpose Scope	*composed of / composes*	**Component** **Performer**
		achieves / achieved by	**Mission**

[2] The relations presented in this figure and the following are not exhaustive but seek to show the primary relations for the topic area.

Table 1 Architecture Definition

Entity Class	Attributes	Relations	Target Classes
	Time Frame[3]		
Component (Type = Family of Systems, Systems Architecture, System of Systems)	See SDG Type: Family of Systems, System Architecture, or System of Systems	*composes / composed of*	**Architecture**
Mission	Description Number	*achieved by / achieves*	**Architecture**
Performer (Type = Operational Architecture)	Abbreviation Cost Description Doc. PUID Latitude Location[4] Longitude Number Operations Purpose Receptions Values Type: Operational Architecture	*composes / composed of*	**Architecture**

2.2 Capture Source Material

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

- **Capability** entity for each source capability statement[5]
- **Document** entity for each source document
- **Mission** entity for each pertinent mission area or description
- **OperationalTask** entity for each operational task from a source such as the Universal Joint Task List (UJTL) or the Mission Essential Task List (METL)[6]
- **Requirement** entity for each source requirement[7]
- **ExternalFile** entity for each source guidance, requirement, mission, or operational task-related table or graphic
- **DefinedTerm** entity 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:

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

- Place any tables and graphics in separate files and reference them in the project repository using **ExternalFile** entities where each entity *augments* the subject entity. The included reports will automatically include these external tables and graphics in the output immediately following the entity 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 entity. For a special term, the term is the name of the entity 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 reports.

Entities from source documents. The entry of source entities into a GENESYS project is accomplished by either entering the data into GENESYS manually or by opening the originating document and using a series of copy and paste commands to copy items out of the original document and pasting the information into GENESYS.

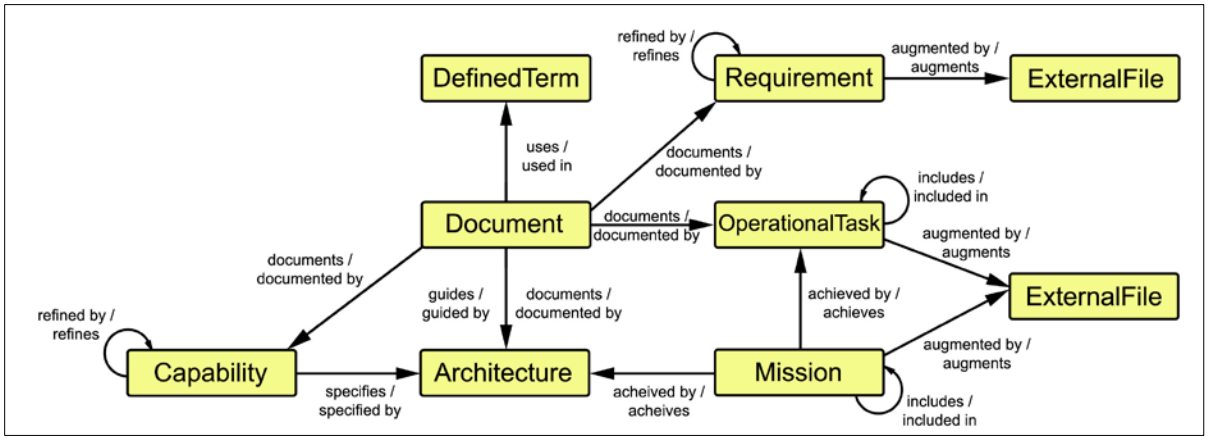

Figure 5 Source Material

Table 2 Source Material

Entity Class	Attributes	Relations	Target Classes
Architecture	See Section 2.1	*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**
		specified by / specifies	**Capability** **Requirement**

Table 2 Source Material

Entity Class	Attributes	Relations	Target Classes
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 / specifies	**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 External File Path Non-Govt. Category Number Revision Number Type	*documents / documented by*[8]	**Architecture** **Capability** **Mission** **Needline** **OperationalActivity** **OperationalItem** **OperationalTask** **Performer** **Requirement** **State**
		uses / used in	**DefinedTerm**

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

Table 2 Source Material

Entity Class	Attributes	Relations	Target Classes
ExternalFile	Description External File Path Number Page Orientation Title Type	*augments / augmented by*[9]	**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**
		includes / included in	**Mission**
OperationalTask	Description Number	*achieves / achieved by*	**Mission**
		augmented by / augments	**ExternalFile**
		documented by / documents	**Document**
		includes / included in	**OperationalTask**
Requirement	Description Doc. PUID Key Performance Parameter	*augmented by / augments*	**ExternalFile**
		documented by / documents	**Document**

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

Table 2 Source Material

Entity Class	Attributes	Relations	Target Classes
	Incentive Performance Parameter[10] Number Origin: Operational Paragraph Number Paragraph Title Rationale Units Value Weight Factor	*refined by / refines*	**Requirement**

2.3 Identify Organizations

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

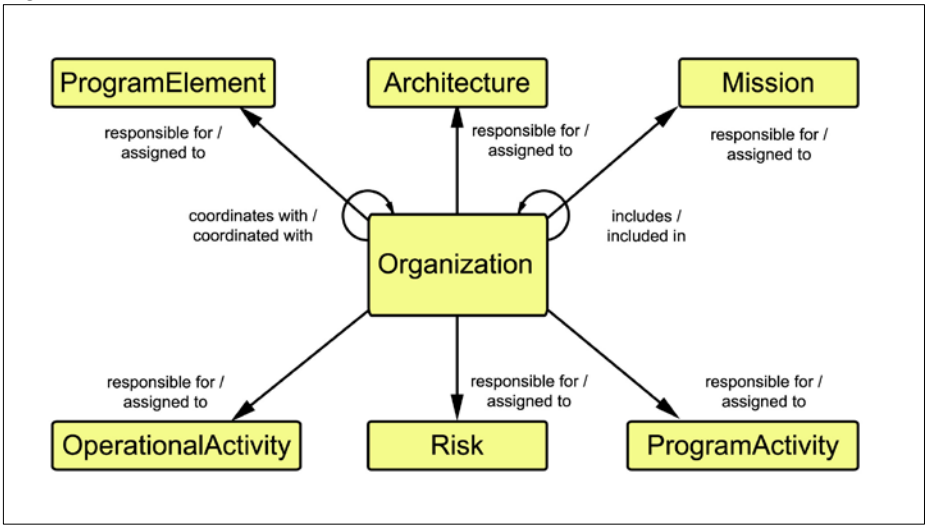

Figure 6 Organizations

Table 3 Organizations

Entity Class	Attributes	Relations	Target Classes
Organization	Abbreviation Description Latitude	*coordinates with / coordinated with*	**Organization**
		includes / included in	**Organization**

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

Table 3 Organizations

Entity Class	Attributes	Relations	Target Classes
	Location Longitude Number Role	*responsible for / assigned to*	**Architecture** **OperationalActivity** **Mission** **ProgramActivity** **ProgramElement** **Risk**

2.4 Define Operational Boundary

Based on an examination of the source, identify the operational boundary and context. To define the boundary, identify each external operational external element with which the architecture must interface. An external operational element (hereafter referred to as an external) is represented as a **Performer** and may identify the operational environment. Create a **Performer** entity representing the context and decompose it into the operational architecture and its externals using the *built from* relation. 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 entities of the **Needline** class. Defining a **Needline** entity establishes that the architecture interacts with an external. Typically, there will be only one **Needline** between the architecture's performers and each external performer.

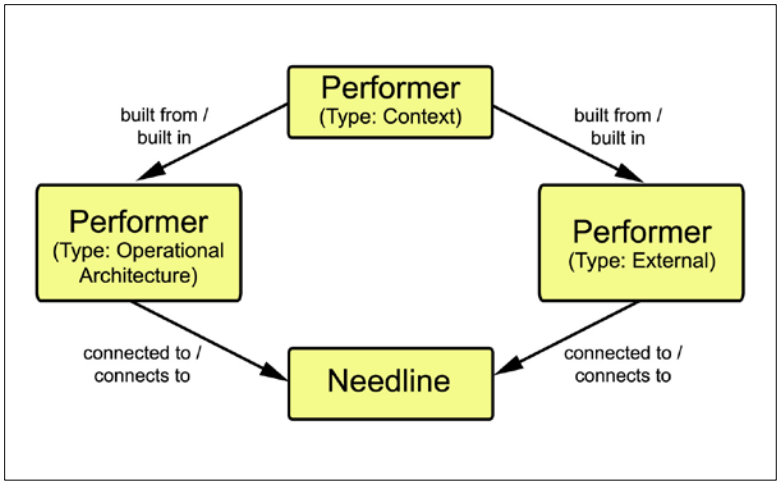

Figure 7 Operational Boundary

Table 4 Operational Boundary

Entity Class	Attributes	Relations	Target Classes
Performer (Type: Context)	See Section 2.1	*built from / built in*	**Performer** (Type: Operational Architecture and External)
Performer	See Section 2.1	*built in / built from*	**Performer**

(Type: External)		*connected to / connects to*	**Needline**
Performer (Type: Operational Architecture)	See Section 2.1	*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 entities in the performer hierarchy in order to differentiate them in GENESYS views such as the Physical Block Diagram and Hierarchy diagrams.

2.5 Classification

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

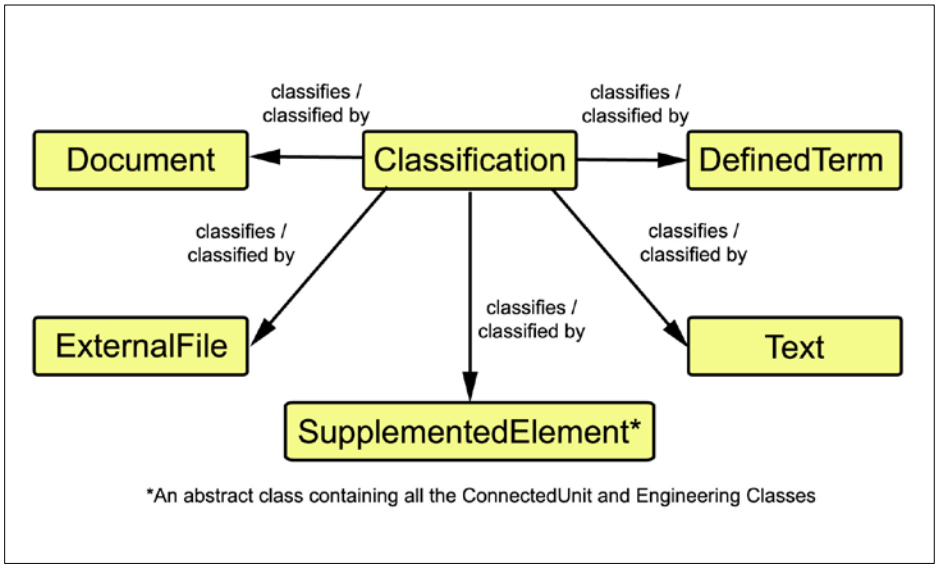

Figure 8 Classification

Table 5 Classification

Entity 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**

3. 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.

3.1 Operational Activity View

Capabilities[11] 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, **OperationalActivities**, 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.

[11] 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,[12] 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.[13] 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** which are executable behavior entities. Each **OperationalActivity** is *allocated to* an entity 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* relation. Logical groupings (taxonomy) of **Capabilities** may be established through the *categorized by* relation with entities within the class **Category**. The context-level **OperationalActivity** is *allocated to* the context-level **Performer** (of Type Context) with the relationship attribute Behavior Type set to "Integrated (Root)."

OperationalActivity Inputs and Outputs. Each **OperationalActivity** within a behavioral view will have input and output **OperationalItem** entities identified. These **OperationalItem** entities are associated with **OperationalActivities** using the relations: *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 2.1), for each level 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* relation. 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** entity (or **OperationalTask** if required) is established using the *achieves* relation. 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 relations are the *specified by* and the *based on* relations. The *specified by* relation identifies constraint or performance requirements that the **OperationalActivity** must satisfy. The *based on* relation 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,** defines the full behavior of the **Performer** from the **Performer**'s perspective, which satisfies both the **Performer**'s external observables and its allocated **Capabilities**.

[12] 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.

[13] Sometimes the external stimulus is not provided upon output because the behavior is internally satisfied within the **Component**.

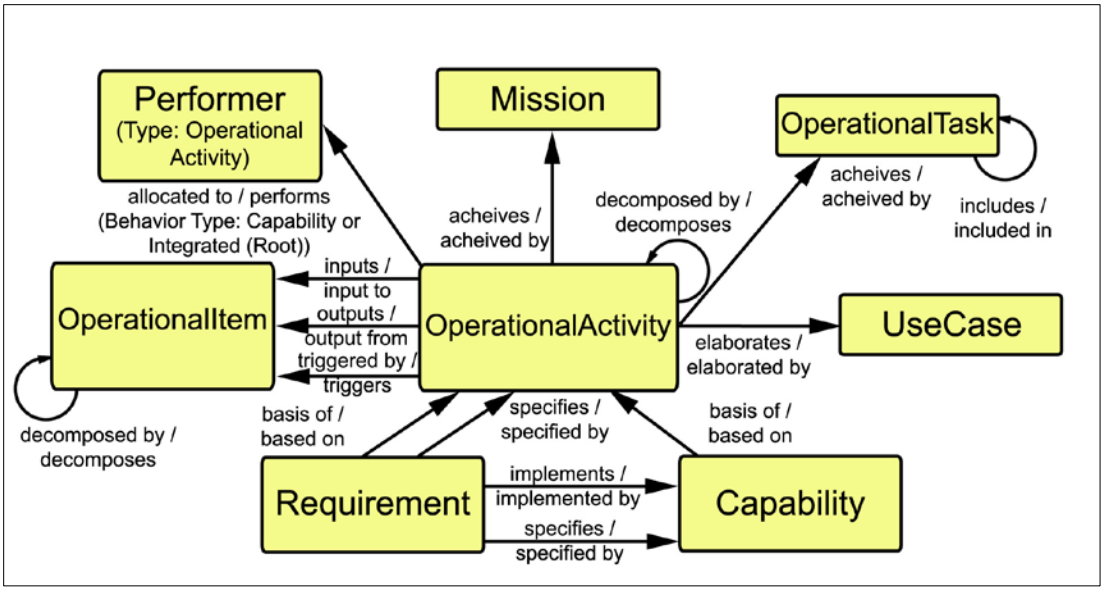

Figure 9 Operational Activity View

Table 6 Operational Activity View

Entity Class	Attributes	Relations	Target Classes
Capability	See Section 2.2	*based on / basis of*	**OperationalActivity**
Mission	See Section 2.2	*achieved by / achieves*	**OperationalActivity**
Performer (Type: Operational Architecture)	See Section 2.1	*performs / allocated to* (Behavior Type: Capability or Integrated (Root))[14]	**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**

[14] 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

Entity Class	Attributes	Relations	Target Classes
		allocated to / performs (Behavior Type: Capability or Integrated (Root))	**Performer**
		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 2.2	*achieved by / achieves*	**OperationalActivity**
		achieves / achieved by	**Mission**
Requirement	See Section 2.2	*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**

3.2 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 Behavior Type 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** entity 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**; this is shown as the **State** *encompassed by* a **Mode**. A **Performer** *exhibits* a **State**, and also *contains* a **Mode**.

Each **State** *incorporates* one or more **OperationalActivities** which specifies behaviors that occur during the execution of the State. Associated with the *incorporates* relation is a Behavior Type attribute. A relationship attribute value of "Entry" indicates behavior that is performed upon entry into the **State**. A value of "Exit" indicates behavior that is performed immediately before exiting the **State**. A value of "Integrated (Root)" indicates behavior that is performed once the "Entry" behavior completes and continues until it finishes or the **State** exits.

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* relation 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**.

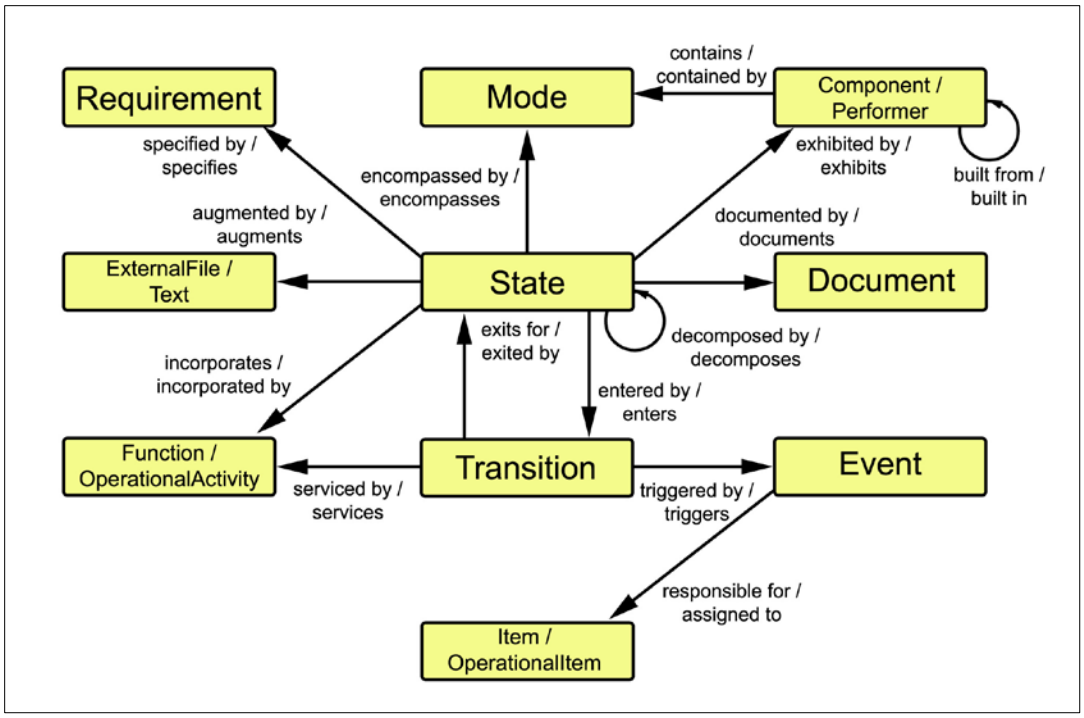

Figure 10 State View

Table 7 State View

Entity Class	Attributes	Relations	Target Classes
Component	See SDG	*contains / contained by*	**Mode**
		exhibits / exhibited by	**State**
Event	Condition Description Doc. PUID Type	*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 Doc. PUID Number	*augmented by / augments*	**ExternalFile** **Text**
		contained by / contains	**Component** **Performer**
		documented by / documents	**Document**

Entity Class	Attributes	Relations	Target Classes
		encompasses / encompassed by	**State**
		impacted by / impacts	**Concern** **Risk**
		specified by / specifies	**Requirement**
OperationalActivity	See Section 3.1	*incorporated by / incorporates*	**State**
		services / serviced by	**Transition**
OperationalItem	See Section 3.1	*assigned to / responsible for*	**Event**
Performer	See Section 2.1	*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**
		exhibited by / exhibits	**Component** **Performer**
		exited by / exits	**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**
		exits / exited by	**State**
		triggered by / triggers	**Event**

4. OPERATIONAL ARCHITECTURE SYNTHESIS

4.1 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 behaviors.

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 **OperationalActivities** are indicated as 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* relation.[15] With all the previous relationships established as described in Section 3.1 for each layer of **Performer** decomposition, 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 3.1, 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.

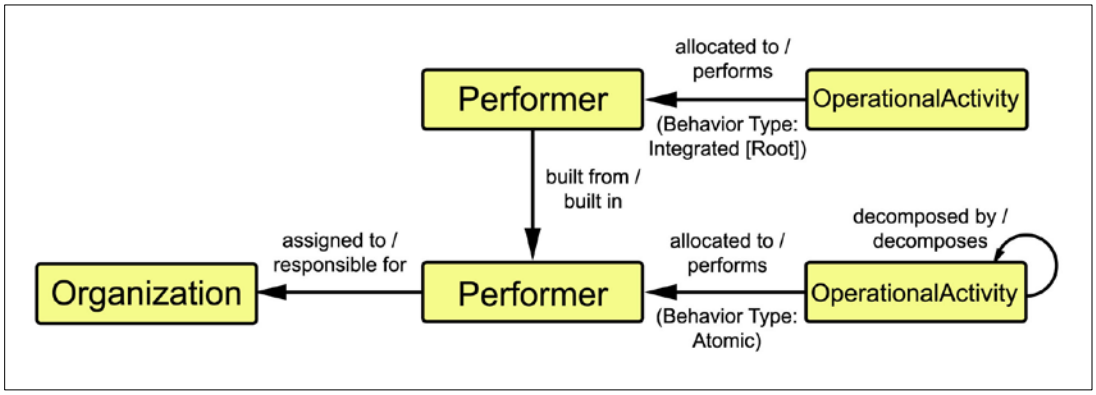

Figure 11 Performer Hierarchy and OperationalActivity Assignment

Table 8 Performer Hierarchy and OperationalActivity Assignment

Entity Class	Attributes	Relationships	Target Classes
Performer	See Section 2.1	*assigned to / responsible for*	**Organization**
		built from / built in	**Performer**
		built in / built from	**Performer**
		performs / allocated to	**OperationalActivity**
OperationalActivity	See Section 3.1	*allocated to / performs*	**Performer**
Organization	See Section 2.3	*responsible for / assigned to*	**Performer**

[15] Organizations, organizational units, roles, etc. are represented as **Organization** entities with a parent-child relation 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 needed between them.

4.2 Refine External Needline Definitions

An external **Needline** entity identifies that the operational architecture communicates in some manner with an external **Performer** (See Section 2.4).[16] **Needlines** are decomposable by means of the *includes* relation. 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**. A Needline should only transfer the lowest level of **OperationalItem**.

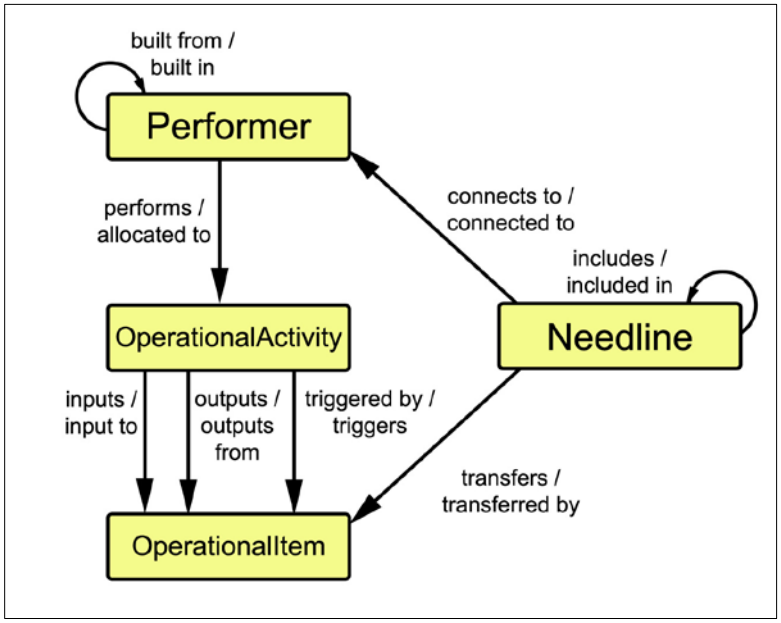

Figure 12 External Needline Definition

Table 9 External Needline Definition

Entity Class	Attributes	Relationships	Target Classes
Needline	See Section 2.4	*connects to / connected to*	**Performer**
		includes / included in	**Needline**
		transfers / transferred by	**OperationalItem**
OperationalItem	See Section 3.1	*transferred by / transfers*	**Needline**
		input to / inputs	**OperationalActivity**
		outputs / output from	**OperationalActivity**
		triggers / triggered by	**OperationalActivity**
Performer	See Section 2.1	*connected to / connects to*	**Needline**

[16] If the external **Performer** is a threat source, then the communication entity 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.

Table 9 External Needline Definition

Entity Class	Attributes	Relationships	Target Classes
		built from / built in	**Performer**

4.3 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** entity class.

Needlines are decomposable by means of the *includes* relation. 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**. A **Needline** should only transfer the lowest layer of **OperationalItem**.

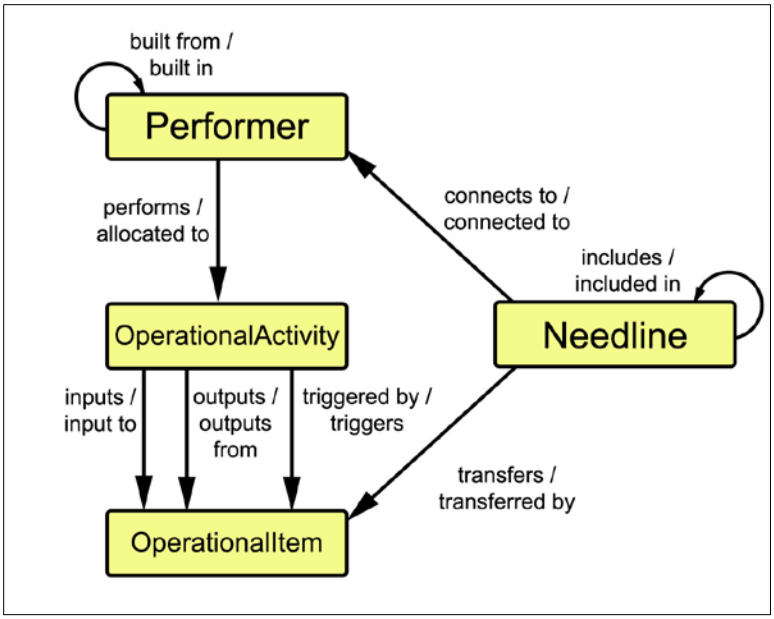

Figure 13 Internal Needline Definitions

Table 10 Internal Needline Definitions

Entity Class	Attributes	Relationships	Target Classes
Needline	See Section 2.4	*connects to / connected to*	**Performer**

Table 10 Internal Needline Definitions

Entity Class	Attributes	Relationships	Target Classes
		includes / included in	**Needline**
		transfers / transferred by	**OperationalItem**
OperationalItem	See Section 3.1	*transferred by / transfers*	**Needline**
		input to / inputs	**OperationalActivity**
		output from / outputs	**OperationalActivity**
		triggers / triggered by	**OperationalActivity**
Performer	See Section 2.1	*connected to / connects to*	**Needline**
		performs / allocated to	**OperationalActivity**

5. OPERATIONAL VIEWPOINT VALIDATION USING THE SIMULATOR

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

6. 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. The systems engineering/architecture activities needed to complete the overall architecture and to interrelate the operational and systems domains are addressed in the following sections.

6.1 Performance Requirements

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

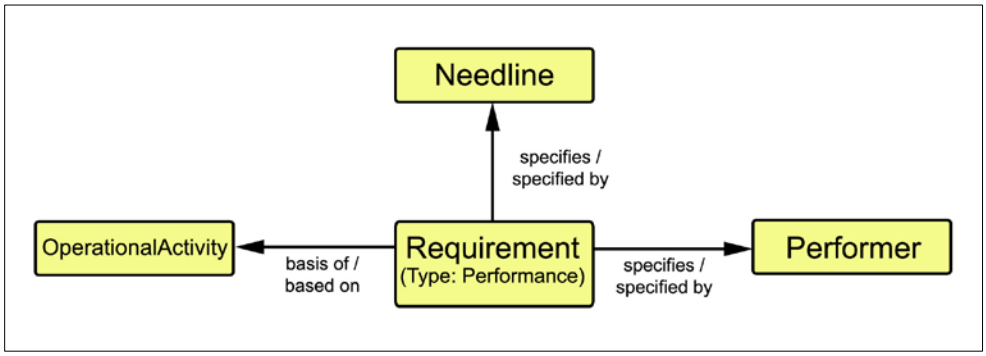

Figure 14 Performance Requirements

Table 11 Performance Requirements

Entity Class	Attributes	Relations	Target Classes
Needline	See Section 2.4	*specified by / specifies*	**Requirement**
OperationalActivity	See Section 3.1	*based on / basis of*	**Requirement**
Performer	See Section 2.1	*specified by / specifies*	**Requirement**
Requirement	See Section 2.2	*basis of / based on*	**OperationalActivity**
		specifies / specified by	**Performer** **Needline**

6.2 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 entities that compose the functionality of a service are collected under a root **Function** via the *decomposed by* relation.

Services are created as a **Component** entity with the Type attribute set to "Service." The Service Type attribute should be set to "Consumer," "Provider," or "Both" as appropriate. The **Component** entity *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**.

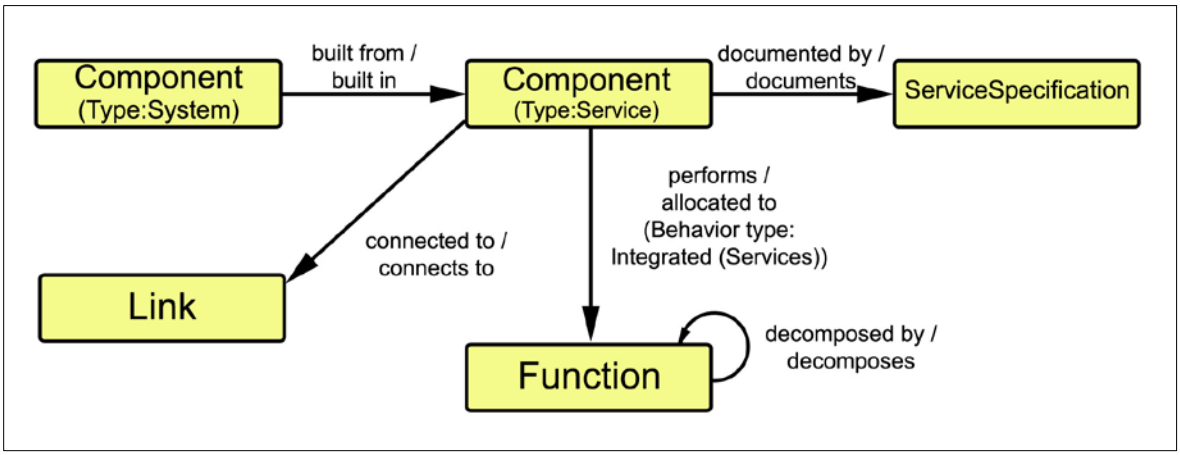

Figure 15 Services

Table 12 Services

Entity Class	Attributes	Relations	Target Classes
Component	See SDG	*built from / built in*	**Component** (Type: Service)
Component	See SDG	*joined to / joins*	**Interface**

Table 12 Services

Entity Class	Attributes	Relations	Target Classes
(Type: Service)	Type: Service	*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	*connects to / connected to*	**Component**
ServiceSpecification	Access Criteria Authentication Mechanism Data Type Effects Information Security Markings Overview Point Of Contact SAP Type Service Access Point Service Version WDSL - Web Services Definition Language	*documents / documented by*	**Component** (Type: Service)

6.3 Requirements Development

Capabilities, OperationalActivities, and **UseCases** serve as sources for system **Requirements**. **Capabilities**, more typically are associated with **OperationalActivities,** 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 relationships between these classes is provide in Figure 6. See the SDG for a description and use of **Requirement** attributes.

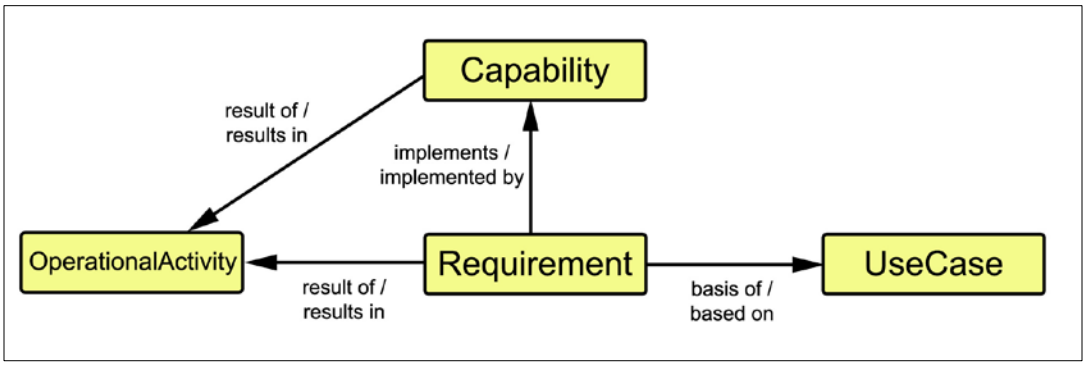

Figure 16 Requirements Development

Table 13 Requirements Development

Entity Class	Attributes	Relations	Target Classes
Capability	See Section 2.2	*implemented by / implements*	**Requirement**
		result of / results in	**OperationalActivity**
OperationalActivity	See Section 3.1	*results in / result of*	**Capability** **Requirement**
Requirement	See Section 2.2	*basis of / based on*	**UseCase**
		implements / implemented by	**Capability**
		result of / results in	**OperationalActivity**
UseCase	See Section 3.1	*based on / basis of*	**Requirement**

6.4 Traceability from Operational Architecture

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

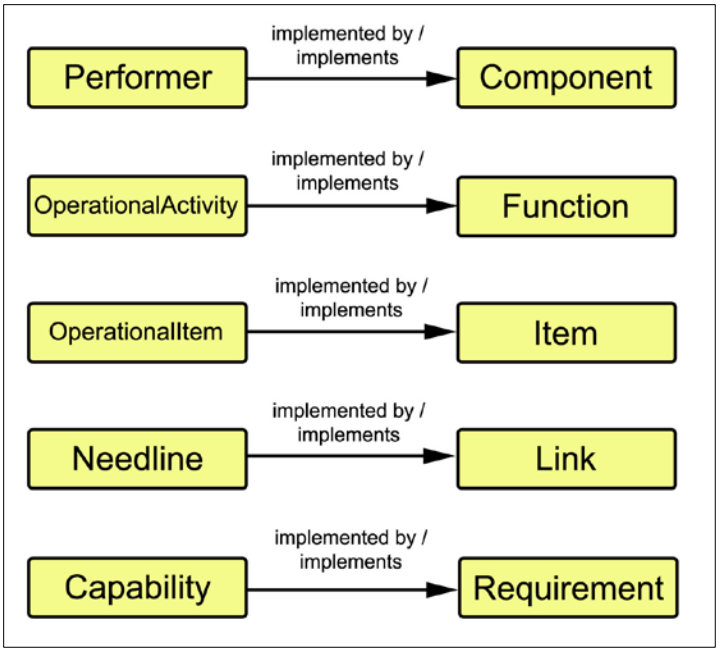

Figure 17 Operational to Systems Traceability

Table 14 Operational to Systems Traceability

Entity Class	Attributes	Relations	Target Classes
Capability	See Section 2.2	*implemented by / implements*	**Requirement**
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 2.4	*implemented by / implements*	**Link**
OperationalActivity	See Section 3.1	*implemented by / implements* (Status: nil, Planned, Partial, or Full)	**Function**
OperationalItem	See Section 3.1	*implemented by / implements*	**Item**
Performer	See Section 2.1	*implemented by / implements*	**Component**

Table 14 Operational to Systems Traceability

Entity Class	Attributes	Relations	Target Classes
Requirement	See SDG	*implements / implemented by*	**Capability**

7. PROGRAM MANAGEMENT ASPECTS

Managing operational 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.

7.1 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 entity within the **ProgramElement** class represents some aspect of the structure of the program or project. These entities are related through the *included in / includes* relation pair. When complete, the resulting hierarchical structure represents the *Work Breakdown Structure* for the program or project. The Type attribute identifies whether the program entity instance is a "Program," "Project," "Work Package," or "Task." The top-most program entity (Type: "Program") *implements* an **Architecture**.[17] *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 entities 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 *supplied by* a **ProgramElement,** which *implements* an **Architecture**. Note: A **Capability** is the *basis of* an **OperationalActivity** (see Section 3.1). A **ProgramElement** also supplies one or more stakeholder deliverables (i.e., an entity of some or all of these classes **Capability**, **Component**, **Document,** or **Performer**).

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

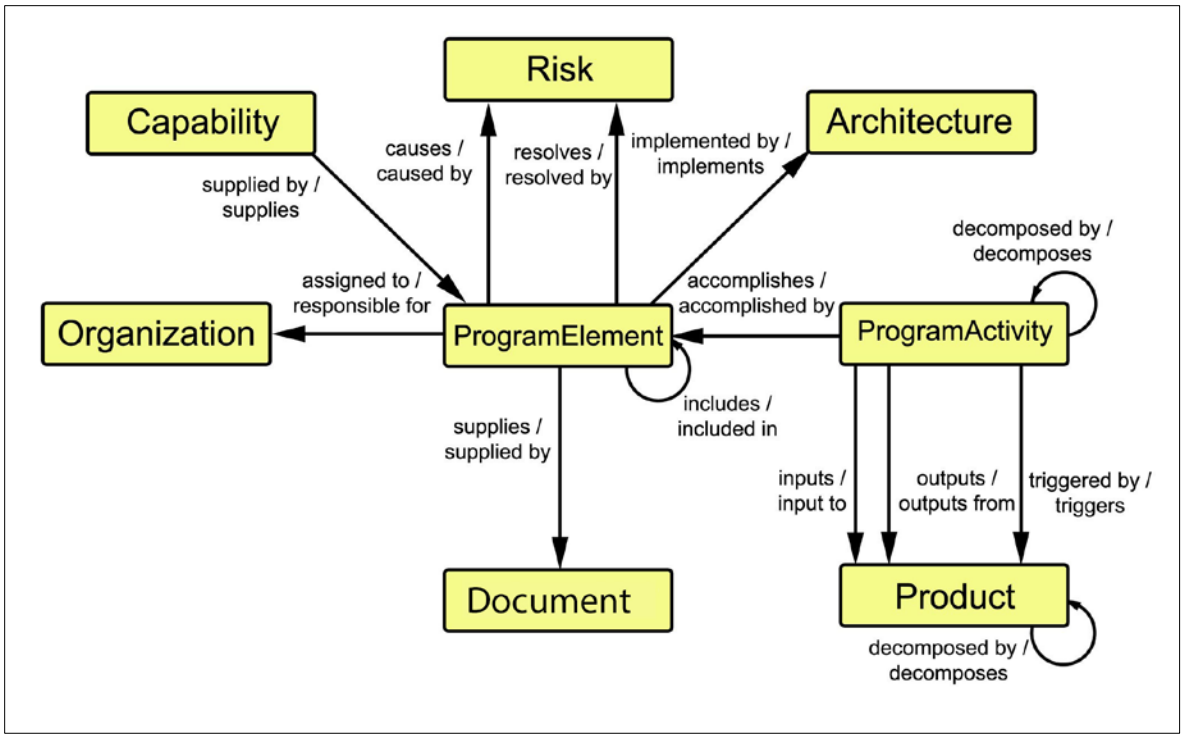

Figure 18 Program Management Basics

Table 15 Program Management Basics

Entity Class	Attributes	Relations	Target Classes
Architecture	See Section 2.1	*implemented by / implements* (Status: nil, Planned, Partial, or Full)	**ProgramElement**
Capability	See Section 2.2	*supplied by / supplies*	**ProgramElement**
Component	See SDG	*supplied by / supplies*	**ProgramElement**
Document	See Section 2.2	*supplied by / supplies*	**ProgramElement**
Performer	See Section 2.1	*supplied by / supplies*	**ProgramElement**
Organization	See Section 2.3	*responsible for / assigned to*	**ProgramElement**
Product	Description Number Size Size Units Type	*decomposed by / decomposes*	**Product**
		input to / inputs	**ProgramActivity**
		output from / outputs	**ProgramActivity**
		triggers / triggered by	**ProgramActivity**

Table 15 Program Management Basics

Entity Class	Attributes	Relations	Target Classes
ProgramActivity	BeginLogic Description Doc. PUID Duration EndLogic ExitLogic Number Timeout Title	*accomplishes / accomplished by*	**ProgramElement**
		decomposed by / decomposes	**ProgramActivity**
		inputs / input to	**Product**
		outputs / output from	**Product**
		Triggered by / triggers	**Product**
ProgramElement	Contract Number Cost Description End Date Labor Hours Non-recurring Cost Start Date Type	*accomplished by / accomplishes*	**ProgramActivity**
		assigned to / responsible for	**Organization**
		augmented by / augments	**ExternalFile**
		causes / caused by	**Risk**
		implements / implemented by	**Architecture**
		includes / included in	**ProgramElement**
		resolves / resolved by	**Risk**
		specified by / specifies	**Requirement**
		supplies / supplied by	**Capability** **Component** **Performer**
Risk	Consequence Likelihood Mitigation Plan Risk Efffective Date Significance Status	*caused by /causes*	**ProgramElement**
		resolved by / resolves	**ProgramElement**

7.2 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 represented by 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** entity 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**. The simulator (see Section 5) 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. The simulator 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** entities identified. These **Product** entities are associated with **ProgramActivities** using the relations: *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** entity is established using the *based on* relation. Traceability to a **ProgramElement** uses the *accomplishes* relation thereby, identifying the **ProgramActivities** that accomplish some aspect of the work breakdown structure.

ProgramActivity traceability from an appropriate **Capability** occurs through the *supplied by* relations to an intermediary **ProgramElement**. The **ProgramElement** *supplies* a **Capability** and is *accomplished by* one or more **ProgramActivities**.

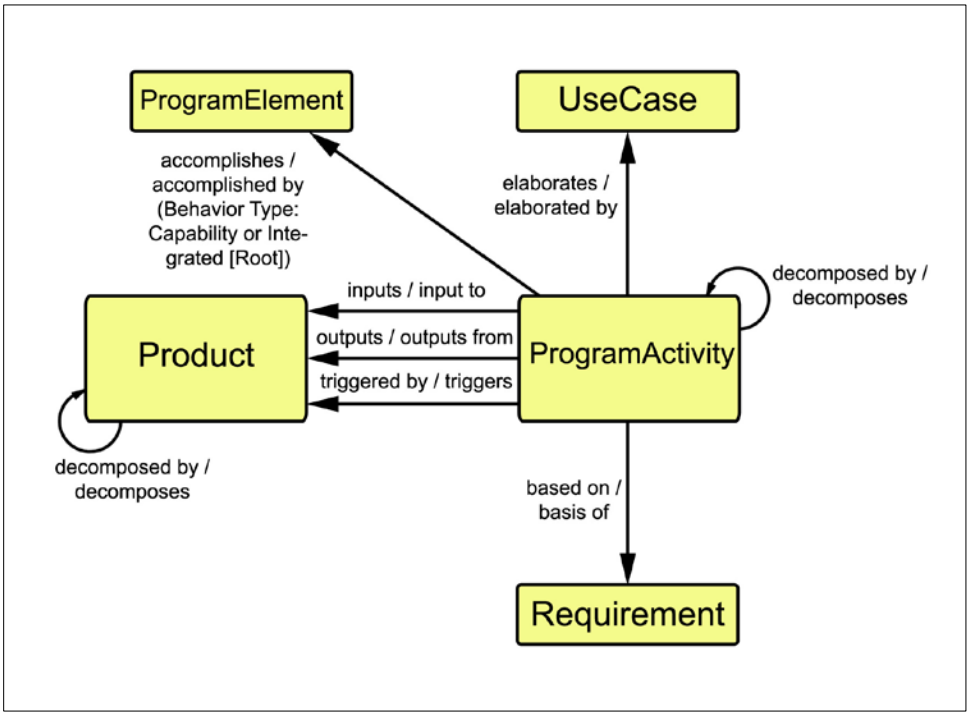

Figure 19 Program Activity View

Table 16 Program Activity View

Entity Class	Attributes	Relations	Target Classes
ProgramActivity	See Section 7.1	*accomplishes / accomplished by*	**ProgramActivity**
		based on / basis of	**Requirement**

Table 16 Program Activity View

Entity Class	Attributes	Relations	Target Classes
		decomposed by / decomposes	**ProgramActivity**
		elaborates / elaborated by	**UseCase**
		inputs / input to	**Product**
		outputs / output from	**Product**
		triggered by	**Product**
ProgramElement	See Section 7.1	*accomplished by / accomplished*	**ProgramActivity**
Product	See Section 7.1	*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**
Requirement	See Section 2.2	*basis of / based on*	**ProgramActivity**
Use Case	See Section 3.1	*elaborated by / elaborates*	**ProgramActivity**

8. DOCUMENTATION—DODAF V2.02 VIEWPOINTS

GENESYS includes a set of reports to output each of the DoDAF v2.02 viewpoints. As appropriate to the particular viewpoint, each viewpoint document contains a standard GENESYS diagram, a table generated from the contents of the repository, or an external file referenced by an **ExternalFile** entity. Because the viewpoints are generated as a result of applying the MBSE process to architecture definition, these reports 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 Reports

Viewpoint	Viewpoint Title	Document 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**.

Table 17 DoDAF v2.02 Viewpoint Reports

Viewpoint	Viewpoint Title	Document Output
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 entities 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** *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 entities used and their attributes and relations stemming from the **Architecture** *composed of* **Component** of Type: "Service," 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

Table 17 DoDAF v2.02 Viewpoint Reports

Viewpoint	Viewpoint Title	Document Output
		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**.
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 Entity 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).**

Table 17 DoDAF v2.02 Viewpoint Reports

Viewpoint	Viewpoint Title	Document Output
SvcV-6	Services Resource Flow Matrix	Summary matrix or full matrix for data exchanges of the children of **Component (s)** of Type: “Service” that *compose*s 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**.
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 entities along with the description of emerging standards and potential impact on current solution entities, 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)** of 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)** of 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	Matrix mapping **Component(s)** of Type: “System” that

Table 17 DoDAF v2.02 Viewpoint Reports

Viewpoint	Viewpoint Title	Document Output
	Systems Traceability Matrix	*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 Description	Hierarchy Diagram of **Components** of Type: “System” that *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 reports, GENESYS provides several other reports 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 FAX: 540.951.8222
Customer Support: support@vitechcorp.com
www.vitechcorp.com