NPS-AM-08-134

Software Hardware Asset Reuse Enterprise (SHARE) Repository Framework Final Report: Component Specification and Ontology

30 September 2008

by

Jean Johnson, Research Assistant, and

Curtis Blais, Research Associate

Graduate School of Engineering & Applied Sciences

Naval Postgraduate School

The research presented in this report was supported by the Acquisition Chair of the Graduate School of Business & Public Policy at the Naval Postgraduate School.

Abstract

In August 2006, Program Executive Officer of Integrated Warfare Systems (PEO-IWS), established the Software Hardware Asset Reuse Enterprise (SHARE) repository to enable the reuse of combat system software and related assets. In July 2007, the Naval Postgraduate School (NPS) was tasked to develop a component specification and ontology for the SHARE repository. A description of SHARE and the requirements for a component specification and ontology supporting this repository are available in Johnson (2007). A vision of the component specification and ontology for the repository framework, a brief survey of initiatives and technologies relevant to desired repository capabilities, a development approach, and initial design are described in Johnson & Blais (2008). The current report provides the initial component specification and ontology for the repository framework, as well as initial information models supporting future implementation of stronger semantic representations of assets and artifacts in the repository. The document provides recommendations for improvements to descriptions of information stored in the repository to enable more effective search, discovery, and use of information. The initial component specification and ontology will help meet near-term repository objectives and will set the foundation for achieving long-term resource discovery objectives in the Global Information Grid context.

Keywords: Software Reuse, Software repository, Component Specification, Ontology, Extensible Markup Language, XML

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgements

The authors would like to thank the sponsor Mr. Nick Guertin, PEO IWS 7B, the SHARE program manager, Ms. Barbara Doyal, and Mr. Mark Wessman for their guidance throughout this research.

THIS PAGE INTENTIONALLY LEFT BLANK

About the Authors

Jean Johnson is a research assistant in the Naval Postgraduate School Systems Engineering Department. She performs research for the PEO IWS SHARE program and coordinates a program to develop Modeling and Simulation curriculum for DoD Acquisition workforce personnel. Previously, Ms. Johnson held various positions supporting NAVSEA's Warfare Systems Engineering Division. She is a US Navy active and reserve veteran and continues her Navy affiliation in the Individual Ready Reserve. Ms. Johnson holds a ME in Operations Research and Systems Analysis and a BS in Applied Mathematics from Old Dominion University. She is currently pursuing her PhD in Software Engineering at NPS.

Jean M. Johnson
Systems Engineering Department
Naval Postgraduate School
Monterey, CA 93943-5000
Tel: 831-656-2956
Fax: (831) 656-3219
E-mail: jmjohnso@nps.edu

Curtis Blais is a research associate in the Naval Postgraduate School Modeling, Virtual Environments, and Simulation (MOVES) Institute. He is contributing to metadata design efforts for the SHARE program, the Department of Defense (DoD) M&S Community of Interest Discovery Metadata Specification, M&S Catalog, and standardized DoD Verification, Validation, and Accreditation (VV&A) documentation, as well as international standardization efforts for the Military Scenario Definition Language and the Coalition Battle Management Language. His research interests include application of Web-based technologies to improve interoperability of C2 systems and M&S systems. Mr. Blais hold BS and MS degrees in Mathematics from the University of Notre Dame and is a PhD candidate in the MOVES curriculum at NPS.

Curtis Blais
Modeling, Virtual Environments and Simulation Institute
Naval Postgraduate School
Monterey, CA 93943-5000
Tel: 831-656-3215
Fax: (831) 656-7599
E-mail: clblais@nps.edu

THIS PAGE INTENTIONALLY LEFT BLANK

NPS-AM-08-134

ACQUISITION RESEARCH
SPONSORED REPORT SERIES

Software Hardware Asset Reuse Enterprise (SHARE) Repository Framework Final Report: Component Specification and Ontology

30 September 2008

by

Jean Johnson, Research Assistant, and

Curtis Blais, Research Associate

Graduate School of Engineering & Applied Sciences

Naval Postgraduate School

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Abbreviations

ADL	Architecture Description Language
ASW	Anti-Submarine Warfare
BFTT	Battle Force Tactical Trainer
CEP WASP	Cooperative Engagement Capability Wrap-Around Simulation Program
CIEL	Common Information Element List
CIO	Chief Information Officer
COAL	Common Operational Activities List
COI	Community of Interest
COM	Component Object Model
CPAN	Comprehensive Perl Archive Network
CSFL	Common Systems Function List
DDG	Guided Missile Destroyer
DDMS	DoD Discovery Metadata Specification
DISA	Defense Information Systems Agency
DL	Description Logic
DMS	Discovery Metadata Specification
DoD	Department of Defense
GIG	Global Information Grid
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDL	Interface Definition Language
IRI	International Resource Identifier
ISO	International Organization for Standardization
IWS	Integrated Warfare Systems
KSLOC	Thousands of Source Lines of Code
LCS	Littoral Combat Ship
LSI	Latent Semantic Indexing
M&S	Modeling and Simulation
MDA	Maritime Domain Awareness; also Model Driven Architecture

MDR	Metadata Registry
MOF	Meta Object Facility
MSC	Modeling and Simulation Community of Interest
NCW	Net-Centric Warfare
NDA	Non-Disclosure Agreement
NPS	Naval Postgraduate School
NSWCDD HSI	Naval Surface Weapons Center Dahlgren Division Human System Integration
OA	Open Architecture
OCL	Object Constraint Language
OMG	Object Management Group
OWL	Web Ontology Language
OWL-S	Web Ontology Language for Services
PEO	Program Executive Office
POR	Program of Record
RAS	Reusable Asset Specification
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
RIF	Rule Interchange Format
SBA	Simulation-Based Acquisition
SHARE	Software Hardware Asset Reuse Enterprise
SIAP	Single Integrated Air Picture
SIPRNET	Secret Internet Protocol Router Network
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SPARQL	SPARQL Protocol and RDF Query Language
SQQ-89	Integrated Surface Ship ASW Combat System (AN/SQQ-89)
SSDS	Ship Self Defense System
TSTS	Total Ship Training System
UDDI	Universal Description, Discovery, and Integration
UML	Unified Modeling Language
URI	Universal Resource Identifier

URL	Universal Resource Locator
USW	Undersea Warfare
W3C	World Wide Web Consortium
WS-BPEL	Web Services Business Process Execution Language
WSDL	Web Services Description Language
WS-I	Web Services Interoperability
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XSLT	Extensible Style sheet Language Transformations

THIS PAGE INTENTIONALLY LEFT BLANK

I. Introduction

A. Background

In August 2006, the Program Executive Officer of Integrated Warfare Systems (PEO IWS) established the Software Hardware Asset Reuse Enterprise (SHARE) repository: a library of combat system software and related assets for use by eligible contractors (both prime contractors and subcontractors) for developing or suggesting improvements to Navy Surface Warfare Systems. The SHARE repository is presently being populated. PEO-IWS encourages all current Navy contractors and potential Navy contractors to register for access to the SHARE library to discover the assets it presently contains, as well as to contribute assets that may be useful to the Navy and its contracting community in the future.[1] An unclassified site provides a mechanism to discover available library materials within SHARE, as those materials are populated. The site also hosts the license agreement and Non-Disclosure Agreement (NDA) required for obtaining library materials. Library materials are provided either online through access to the appropriate portion of the SHARE web site (classified or unclassified) or via delivery of physical media. The registration process for the classified portion of the site over SIPRNET[2] is the same as the unclassified portion above, except no digital certificate is required. The most recent SHARE repository update (Version 1.3) incorporates several new enhancements: updated metadata and an improved asset submission process. A more complete description of SHARE and the requirements for a semantic framework consisting of a component (repository asset) specification and ontology that will support this repository are available in Johnson (2007).

[1] Organizations interested in registering for access to the library should visit and complete an online registration form at https://viewnet.nswc.navy.mil

[2] https://viewnet.nswcdd.navy.smil.mil

B. Scope

The Naval Postgraduate School (NPS) is tasked to develop an initial component specification and ontology for the SHARE software[3] repository. The component specification will describe the artifacts contained in the repository in sufficient detail to aid a repository user in determining if the artifact is worth retrieving. The ontology will provide contextual semantics describing relationships among items in the repository to aid in associating artifacts with users' needs. The component specification and ontology will comprise a rich structural and semantic framework for SHARE that will enable multiple kinds of search and discovery techniques. The goal is to enable the development of different tools to improve the usefulness of SHARE.

C. Purpose

The purpose of this report is to extend the description of the intended repository framework outlined in Johnson & Blais (2008) by providing updated specification of the repository framework component specification and ontology. Here, we provide initial information models supporting future implementation of stronger semantic representations of assets and artifacts in the repository. The document provides recommendations for improvements to descriptions of assets and artifacts stored in the repository to enable more effective search, discovery, and use to meet near-term objectives and to set the foundation for achieving long-term objectives for resource discovery in the future Global Information Grid.

[3] While the SHARE repository is intended for information on both hardware and software assets, this initial tasking is limited to a software (code and documentation) asset scope. The goal of the research, however, is for technical approaches that are developed for software to be readily adaptable to descriptions of hardware assets in the repository.

II. Conceptual Vision for the Software Repository Framework

A. Introduction

This section summarizes the rationale and vision for the SHARE repository framework and describes the major components that are specified in this report for implementation.

B. Rationale for an Enriched Semantic Framework

Current software repositories tend to be organized to support keyword searches over broad categories of software types. As an example, the popular online repository SourceForge enables essentially two search types. First, users can browse the repository by clicking through categorizations of different types of software and can then refine the search by filtering for different program aspects, such as specific programming language or operating system. Second, users can perform a keyword search over the metadata within a particular category.

In addition to these typical types of searches, we envision a graphical user interface that enables navigation of repository assets depending on users' interests. This requires an interface that allows users to project their context onto the search mechanisms. In other words, the users bring particular information needs and goals based on the problem they are trying to solve. For example, users may seek particular functionality best obtained through a functional organization of the information in the repository; they may seek particular artifacts best obtained through a document resource organization of the information; or, they may seek information on certain testing methodologies that have been applied so that a work activity organization of the information would best apply. The challenge in designing the framework for the software repository is devising initial sets of taxonomic descriptions of the assets while creating flexibility for future introduction of additional and diverse organizational views (e.g., profiles or templates) of the information as users' needs and repository utility grow.

There are many types of searches that could be implemented based on the proposed repository framework. One type of search that we envision is a "point and click" graphical navigation of repository artifacts that allows users to navigate to and select artifacts based on various relationships to an item of interest. As discussed in the previous reports, various navigation and visualization techniques are possible if relationships among assets and artifacts are recorded. A second type of search that could be implemented is a semantic-matching, document-based search. A related ongoing NPS research project titled ReSEARCH is investigating methods for improving consistent semantic interpretation of requirements documents to improve searches based on text. Formalized semantic descriptions in the SHARE component specification and ontology will further enhance ReSEARCH capabilities to produce highly relevant search findings for users of the SHARE repository. A third type of search of interest is based on a user-constructed model of the problem the user is trying to solve. The user interface for the repository can assist users in building the model of a desired system architecture using a standardized representation scheme. The search can then return possible existing solutions for portions of the system and can identify likely gaps. These concepts are described in more detail in Johnson & Blais (2008).

C. User Goals

The potential goals that users may have when coming to the SHARE repository are based on the lifecycle phase or activity the users are performing at the time of search. We call these "goals" since they do not have the detail normally associated with viable (testable) use cases but could become the basis for creation of such use cases. These goals were used to guide our development of the repository framework and include:

- Concept development
  - Users seek to investigate current systems' capabilities and compare them to new desired capabilities.
  - Users seek to develop an operational model of a system (a model with "holes") and manipulate the model with tools to stimulate creativity towards new types of solutions to operational problems.
- Requirements
  - Users seek to reuse existing requirements, where the proposed system meets existing capabilities.
  - Users seek to generate a draft requirements specification from existing repository contents.
- Design
  - Users seek to search for existing components that may satisfy portions of the requirements.
  - Users seek to retrieve design patterns for common problems.
  - Users seek to reason about a system's architecture including the ability to compare and evaluate possible solution compositions, investigate an architecture's ability to satisfy quality attributes or non-functional requirements (e.g., interoperability, safety, performance, etc.), and investigate an architecture's ability to satisfy functional requirements.
- Implementation
  - Users seek to use existing repository artifacts to help refine a system model towards implementation.
  - Users seek to auto-generate a system based on a desired architecture. The infrastructure provided by this specification and supporting tools will be geared to support composition even if it cannot be fully automated at this time. More likely, a sort of cookbook or wizard to facilitate integration through an interface for assisting humans in making decisions during the composition process will be enabled. At a minimum, the component specification will include useful information for manually constructing the composite system from components.
- Test
  - Users seek to auto-generate testing requirements based on existing repository test information.

    - Users seek to develop test cases and scenarios based on operational models constructed in the concept development phase.
    - Users seek to reuse pre-existing testing requirements based on systems satisfying portions of the operational model or system design.
    - Users seek to reuse tools, environment models, and methodologies for testing.
    - Users seek to investigate the testing pedigree for items in the repository that are candidates for reuse.
- Maintenance
    - Users seek to reuse maintenance plans, implementation of software product line principles for system evolution, and software reuse in reengineering efforts. These goals are outside the scope of this research but are recommended for future research.

In section VI we will investigate two possible user search scenarios based on different lifecycle phases and their goals. The scenarios will be used to demonstrate the utility of each piece of the recommended repository framework.

D. SHARE Repository Framework

To enable the types of tools we envision, we must create a richer semantic framework for the repository. Johnson & Blais (2008) proposed two major aspects for this framework: a component specification and ontology. The component specification is a description or model of the items in the repository and consists of two parts: metadata and software behavior representation. The ontology describes concepts and relationships to create various perspectives or contexts for examining the contents of the repository. These aspects of the framework are discussed further below.

1. Component Specification: Metadata

The metadata for each artifact should incorporate all necessary data for discovery and implementation. The metadata will aid repository users in determining if the item is suited for their use and will provide information about how to use the asset when it is retrieved. We refer to this as “standard” or “typical” metadata since there are many existing examples of metadata we can use to develop the metadata for SHARE. The current and recommended SHARE metadata sets are described in Section III.

2. Component Specification: Software Behavior

The metadata for many current repositories, such as those described earlier, fail to capture a searchable representation of the behavior of the items outside general categories of functionality (e.g., Archiving Compression Conversion, Control Flow Utilities, Graphics, Security) and text-based search of code descriptions. Unlike current practice, the SHARE component specification will consist of both typical metadata and a behavioral model of the component. Since this piece of the component specification is not commonly incorporated into repositories in a standardized manner, we feel it is a specific focus area to identify the appropriate representation mechanisms for software behavior in the repository context. Our recommended approach to incorporating software behavior into the SHARE metadata is presented in Section IV.

3. Ontology of Framework Relationships

The framework ontology presented in section V includes descriptions of the relationships of the components to form a contextual model of the repository items.[4] These relationships may include the component’s use/role in existing systems, its mapping to reference or domain architectures, and its utility in various software development lifecycle phases. Contextual information about the artifact can be

[4] Throughout the document, we will use *ontology* as a general term for describing concepts and relationships among concepts, with *taxonomy* as a special case in which the classes in the ontology are related by a single property, such as “is-a” or “has-a.”

exploited to enable sophisticated search and discovery methods that more closely match recommended retrieval items to a user's problem context.

E. Summary

In developing this framework for the SHARE repository that will enable richer search capabilities to support the achievement of individual repository user goals, we focus on three areas:

1. "Typical" metadata for artifacts
2. A suitable representation of software behavior
3. Framework relationships (ontology)

The following three sections describe the approach we used to address each of these areas and their resulting products. We will also use two use cases reflecting user activities in the requirements and design phases of a project as a demonstration of how the information provided by each product will enable the accomplishment of various user goals.

III. SHARE Metadata

A. Introduction

An initial list of required asset information has been developed by the SHARE Program Office at Naval Surface Warfare Center, Dahlgren, VA. We began our metadata development effort by creating an Extensible Markup Language (XML) Schema for this metadata set and then enhanced the schema based on a more current “wizard” that leads a user through the SHARE asset information entry process. This “as-is” metadata set is provided as Appendix A and described in Section B below.

After careful analysis of this initial schema, as well as known metadata examples found in existing software repositories, we began to modify the schema by both reorganizing the data and complementing the fields with information we believe should be included. We also incorporated the necessary information to place each artifact in the appropriate context based on the ontology development. Finally, we evaluated the schema against the minimum requirements of the DoD Discovery Metadata Specification (Deputy Assistant Secretary of Defense 2007) to promote future exposure of SHARE contents across the DoD Enterprise. The resulting recommended schema is provided as Appendix B and described in section C below.

B. Current Content Description

At the time of this writing, the authors were not aware of any metadata description of SHARE assets and artifacts using the Extensible Markup Language (XML) Schema language.[5] As a starting point for creating structured descriptions of the SHARE data, we designed an XML Schema following the organization and

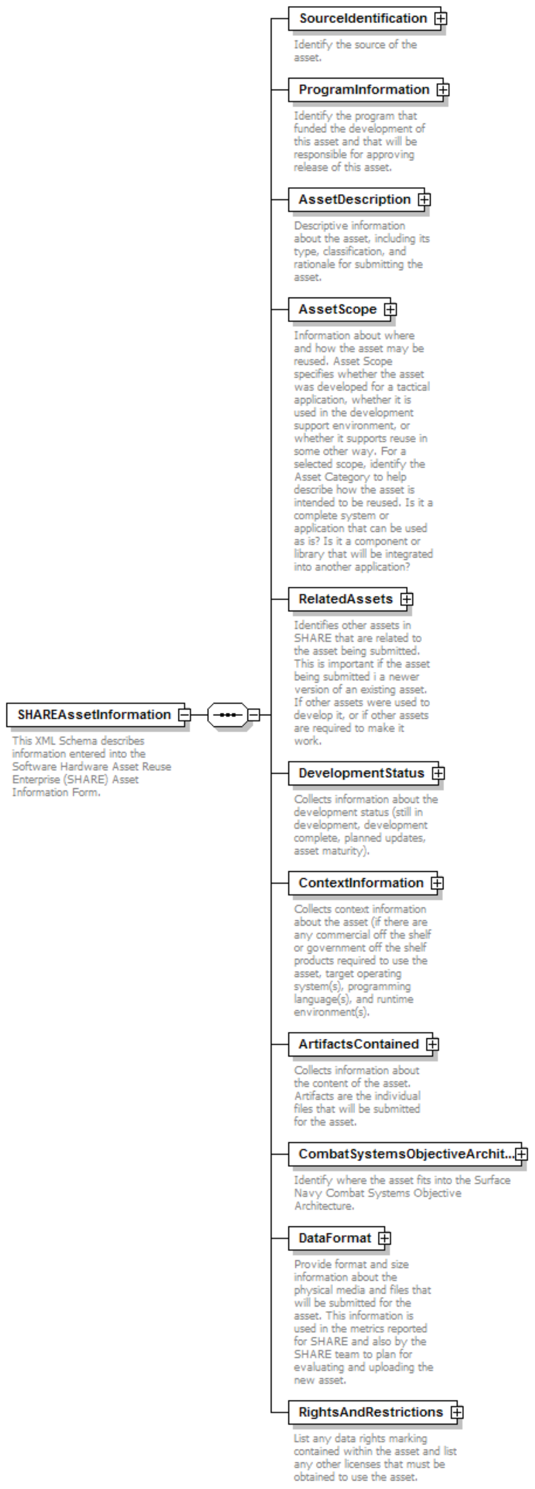
[5] Refer to http://www.w3.org/XML/Schema for full description and specification of the XML Schema language.

content of the SHARE Asset Information Form and the user-entry wizard used to collect information about an asset being submitted. The top-level structure of the XML representation of SHARE Asset Information is shown in Figure 1[6] The root element of the XML structure is called *SHAREAssetInformation*. There is a one-to-one correspondence between major elements in the XML structure (child elements of the root element) with the major steps in the user entry wizard:

- Source Identification (wizard step 2[7])
- Program Information (wizard step 3)
- Asset Description (wizard step 4)
- Asset Scope (wizard step 5)
- Related Assets (wizard step 6)
- Development Status (wizard step 7)
- Context Information (wizard step 8)
- Artifacts Contained in the Asset (wizard step 9)
- Architecture Domain (wizard step 10)
- Data Format (wizard step 11)
- Rights and Restrictions (wizard step 12)

[6] Diagrams of the XML structures have been generated by Altova XML-Spy. The product is available at http://www.altova.com/products/xmlspy/xml_editor.html. Altova offers a free 30-day license for trial use of the product. The Altova presentation of elements incorporates a plus (+) symbol on the right edge of a box to indicate that the element contains subelements. This may be useful for the reader to follow the figures presented in this report since in most cases, the expansion of an element with the plus (+) symbol will be shown in a later figure.

[7] Note: Step 1 of the SHARE Asset Information Form wizard is a basic start-up step with no associated data entry.



SHAREAssetInformation: This XML Schema describes information entered into the Software Hardware Asset Reuse Enterprise (SHARE) Asset Information Form.

SourceIdentification: Identify the source of the asset.

ProgramInformation: Identify the program that funded the development of this asset and that will be responsible for approving release of this asset.

AssetDescription: Descriptive information about the asset, including its type, classification, and rationale for submitting the asset.

AssetScope: Information about where and how the asset may be reused. Asset Scope specifies whether the asset was developed for a tactical application, whether it is used in the development support environment, or whether it supports reuse in some other way. For a selected scope, identify the Asset Category to help describe how the asset is intended to be reused. Is it a complete system or application that can be used as is? Is it a component or library that will be integrated into another application?

RelatedAssets: Identifies other assets in SHARE that are related to the asset being submitted. This is important if the asset being submitted i a newer version of an existing asset. If other assets were used to develop it, or if other assets are required to make it work.

DevelopmentStatus: Collects information about the development status (still in development, development complete, planned updates, asset maturity).

ContextInformation: Collects context information about the asset (if there are any commercial off the shelf or government off the shelf products required to use the asset, target operating system(s), programming language(s), and runtime environment(s).

ArtifactsContained: Collects information about the content of the asset. Artifacts are the individual files that will be submitted for the asset.

CombatSystemsObjectiveArchit...: Identify where the asset fits into the Surface Navy Combat Systems Objective Architecture.

DataFormat: Provide format and size information about the physical media and files that will be submitted for the asset. This information is used in the metrics reported for SHARE and also by the SHARE team to plan for evaluating and uploading the new asset.

RightsAndRestrictions: List any data rights marking contained within the asset and list any other licenses that must be obtained to use the asset.

Figure 1. Top-Level Structure of an XML Representation of SHARE Asset Information

In the following subsections, we provide a brief description of the structure and content of each of these elements. The intent is neither to provide a user's guide to the data entry wizard nor to present a complete description of the full depth of the XML schema; rather, it is the intent to provide an overview of the structure. The full schema is provided in Appendix A for further reference.

1. Source Identification

The *SourceIdentification* element contains information about the person and organization submitting the asset. The structure of the *SourceIdentification* element is shown in Figure 2. The *Contributor* element contains name and contact information on the person entering the asset information (Figure 3).[8] The *Organization* element provides the name of the organization submitting the asset and additional information, depending on whether it is a governmental or contractor organization (Figure 4 and Figure 5).

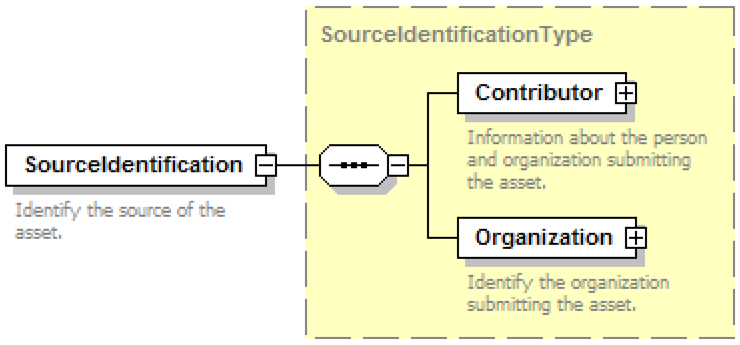

SourceIdentificationType
Contributor
Information about the person and organization submitting the asset.
SourceIdentification
Identify the source of the asset.
Organization
Identify the organization submitting the asset.

Figure 2. SourceIdentification Element

For governmental organizations, the wizard prompts the user to identify any applicable government patents that apply to the asset being submitted (Figure 6).

[8] In the figure, boxes in solid lines represent child elements that are required entries; dashed lines indicate optional entries. The Address and Notification child elements have subordinate structure, indicated by the "+" sign on the right side edge of the boxes.

Child element *ApplicableGovernmentPatents* is a container holding one or more *Patent* elements as needed.

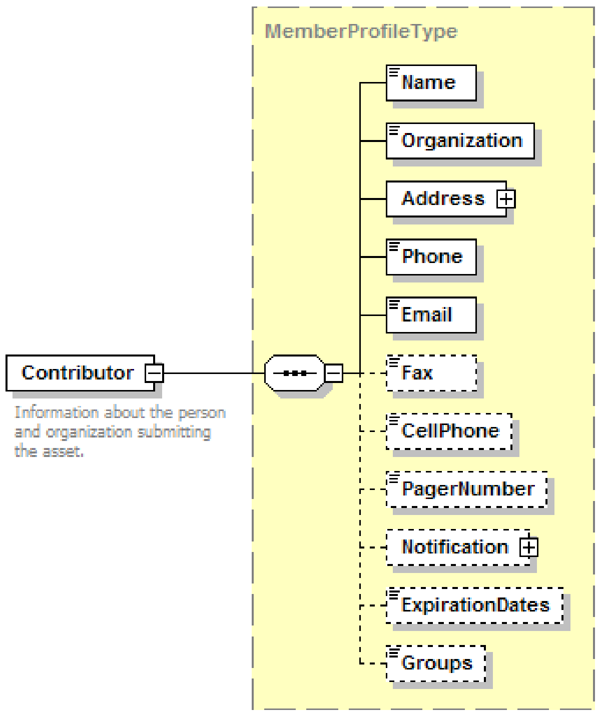


Figure 3. Contributor Element

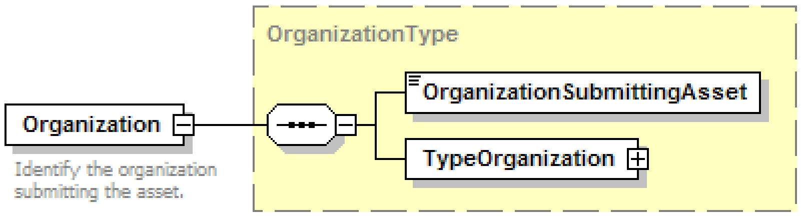

OrganizationType
OrganizationSubmittingAsset
TypeOrganization
Organization
Identify the organization submitting the asset.

Figure 4. Organization Element

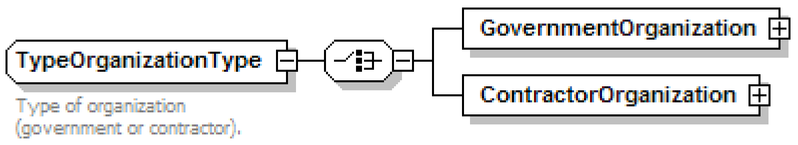


Figure 5. Selection of Government or Contractor Type of Organization

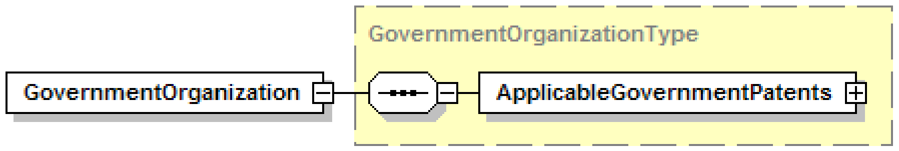


Figure 6. Entry of Applicable Patent Information for a Government Organization

For a contractor organization, the wizard prompts the user to enter a contract number (which the asset was developed under), how the asset was delivered to the government, and name and contact information (organization and address) on a cognizant Contracting Officer (Figure 7).

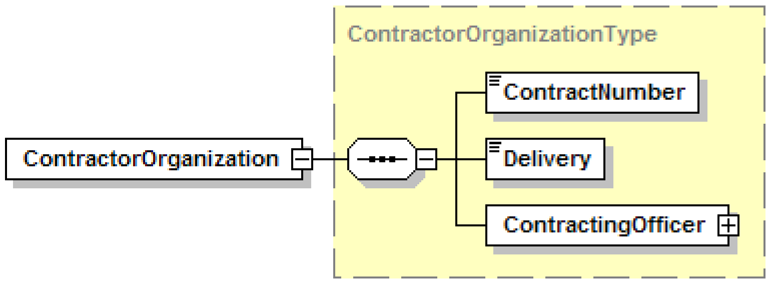


Figure 7. ContractorOrganization Element

2. Program Information

The *ProgramInformation* element (Figure 8) identifies the program that developed the asset. The user can select from a list of previously identified programs or can enter a new program name, in which case the user is prompted by the wizard to enter program manager name and contact information for the identified program.

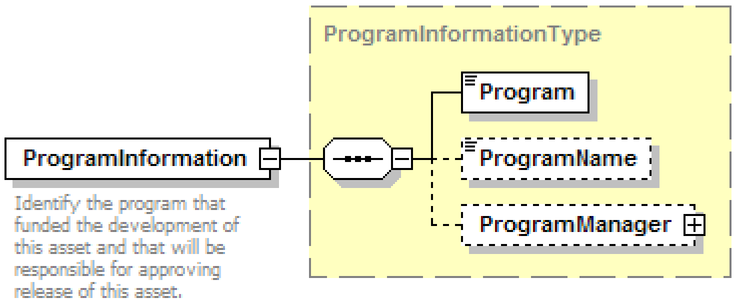

ProgramInformationType
Program
ProgramName
ProgramManager
ProgramInformation
Identify the program that funded the development of this asset and that will be responsible for approving release of this asset.

Figure 8. ProgramInformation Element

3. Asset Description

The *AssetDescription* element (Figure 9) contains information about the asset being submitted to the repository, including its name, type (code or documentation), description, version, date (optional), classification information, and rationale. Classification information (Figure 10) includes the classification level (e.g., C, S), export control statement, distribution statement (e.g., distribution statement A), and, for classified assets, the Security Classification Guide ID number.

It is important to note that the Security Classification Guide ID number is optional in the XML structure but is actually required if a classified asset is entered into the system. The entry wizard can easily enforce this “business rule”; it is more difficult to express the combination in the XML schema, although a number of approaches are possible. Other schema languages, such as Schematron, or ontologies, such as the constructs described later in this document, can be used to convey such conditions more effectively than the XML Schema language alone. Further work in this area can be done following programmatic decisions about use of metadata in the SHARE repository.

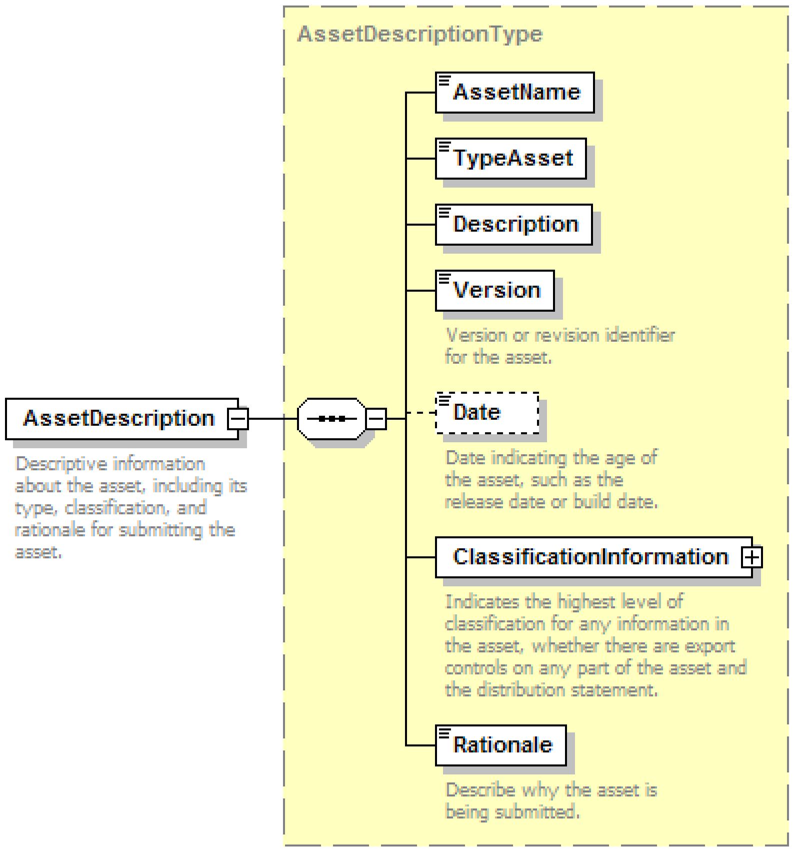

AssetDescriptionType
AssetName
TypeAsset
Description
Version
Version or revision identifier for the asset.
Date
Date indicating the age of the asset, such as the release date or build date.
ClassificationInformation
Indicates the highest level of classification for any information in the asset, whether there are export controls on any part of the asset and the distribution statement.
Rationale
Describe why the asset is being submitted.
AssetDescription
Descriptive information about the asset, including its type, classification, and rationale for submitting the asset.

Figure 9. AssetDescription Element

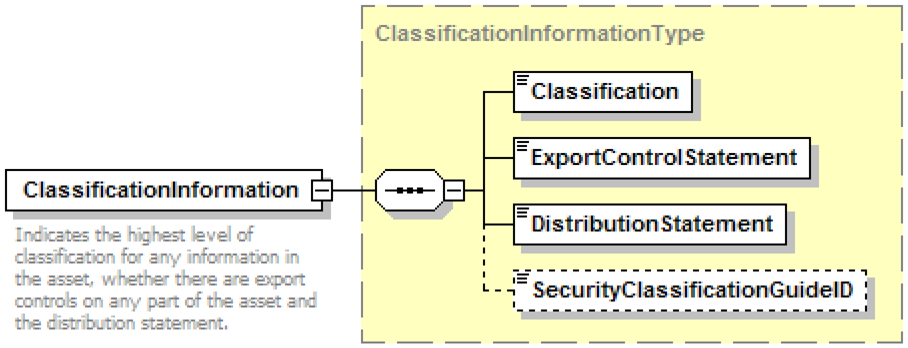

ClassificationInformationType
Classification
ExportControlStatement
DistributionStatement
SecurityClassificationGuideID
ClassificationInformation
Indicates the highest level of classification for any information in the asset, whether there are export controls on any part of the asset and the distribution statement.

Figure 10. ClassificationInformation Element

4. Asset Scope

The *AssetScope* element (Figure 11) contains information specifying whether the asset was developed for a tactical application, whether it is used in the development support environment, or whether it supports reuse in some other way. For each of these selections, the wizard provides a list of asset categories. The XML structure uses the element tag (*TacticalApplication*, *DevelopmentSupport*, *OtherScope*, *UnknownScope*) to indicate the first level of categorization and the value of that element to identify the second level of categorization (e.g., possible values of the TacticalApplication element include “System,” “Application Program,” “Package,” “System Service,” “Component,” “Library,” and “Module/Code Fragment”).

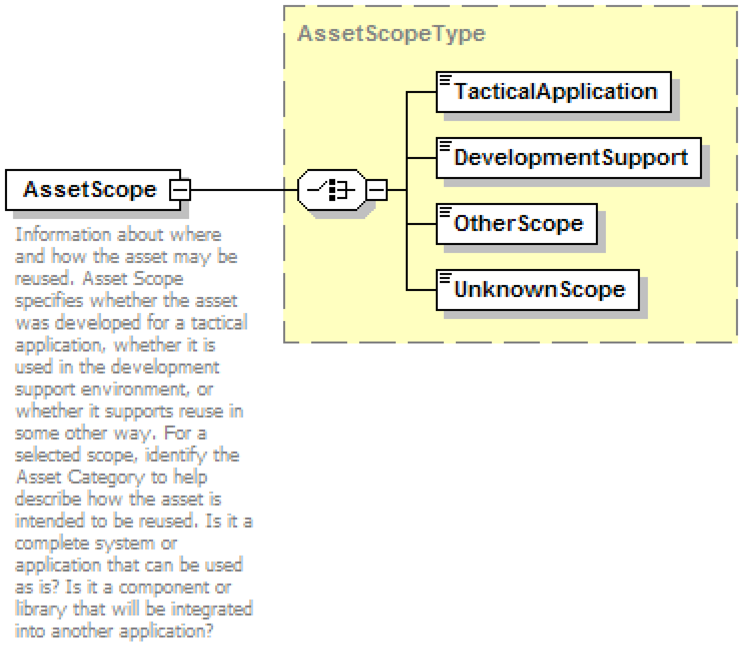

AssetScopeType
TacticalApplication
DevelopmentSupport
OtherScope
UnknownScope
AssetScope
Information about where and how the asset may be reused. Asset Scope specifies whether the asset was developed for a tactical application, whether it is used in the development support environment, or whether it supports reuse in some other way. For a selected scope, identify the Asset Category to help describe how the asset is intended to be reused. Is it a complete system or application that can be used as is? Is it a component or library that will be integrated into another application?

Figure 11. AssetScope Element

5. Related Assets

The *RelatedAssets* element is a container holding one or more *RelatedAsset* elements. This structure identifies other assets in SHARE that are related to the asset being submitted. For an individual *RelatedAsset* element (Figure 12), the XML schema design approach is similar to that described above for the *AssetScope* element; after identifying the related asset by name (the name or a reference to the selected asset would be stored in the XML file conforming to this schema), the user selects a type of relationship, identified here as *CreationRelationship* (Newer Version, Variant Of, Extracted From, or Derived From), *WorkingRelationship* (Dependent Upon, Needed By, or Interfaces With), *GenerationRelationship* (Older Version, Extraction Source, or Derivation Source), or *OtherRelationship* (Similar To, Contained Within, or Contains). The content of each of the child elements identifies the type of relationship (one section of the related values is shown above).

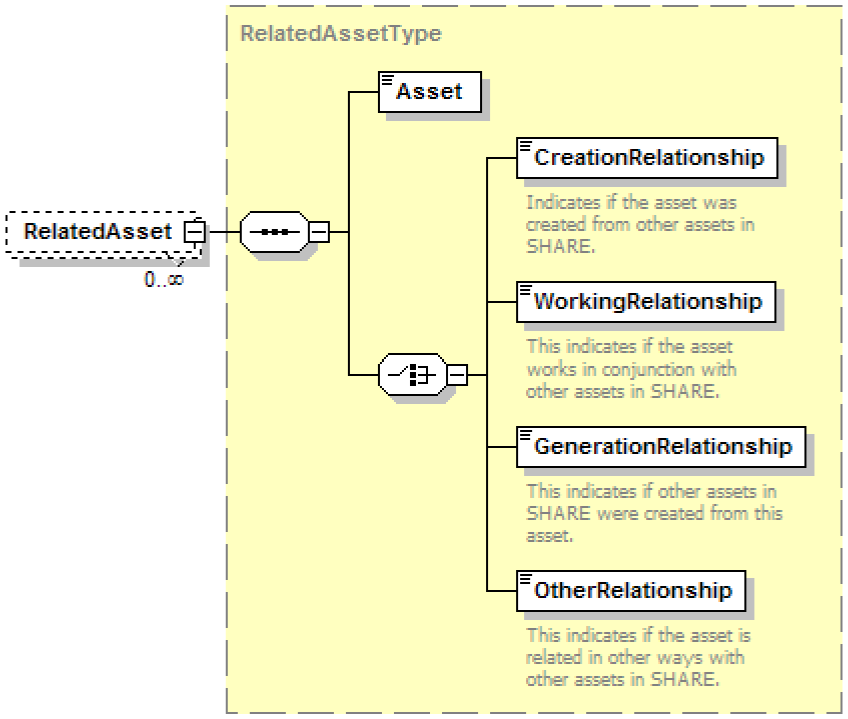

RelatedAssetType
RelatedAsset
0..∞
Asset
CreationRelationship
Indicates if the asset was created from other assets in SHARE.
WorkingRelationship
This indicates if the asset works in conjunction with other assets in SHARE.
GenerationRelationship
This indicates if other assets in SHARE were created from this asset.
OtherRelationship
This indicates if the asset is related in other ways with other assets in SHARE.

Figure 12. RelatedAsset Element

6. Development Status

The *DevelopmentStatus* structure (Figure 13) identifies the development status (In Development or Development Completed) and describes any planned updates and the maturity of the asset.

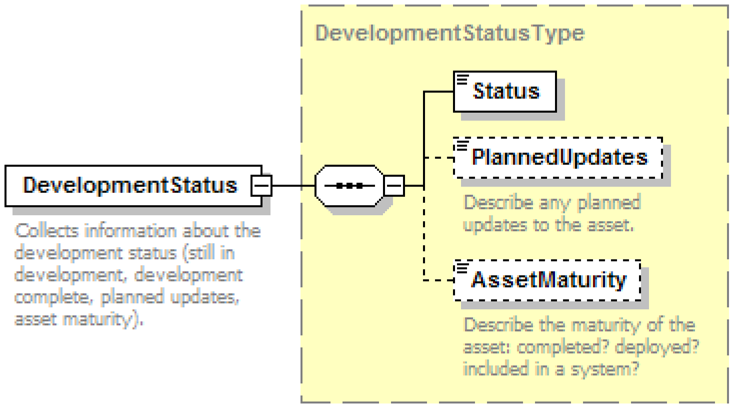

DevelopmentStatusType
Status
PlannedUpdates
Describe any planned updates to the asset.
AssetMaturity
Describe the maturity of the asset: completed? deployed? included in a system?
DevelopmentStatus
Collects information about the development status (still in development, development complete, planned updates, asset maturity).

Figure 13. DevelopmentStatus Element

7. Context Information

The *ContextInformation* element (Figure 14) provides information about the operational context needed for the asset, including dependencies on Commercial Off-the-Shelf Software (COTS) or Government Off-the-Shelf Software (GOTS), target operating system(s), programming language(s), and runtime environment(s). Each child element of *ContextInformation* is a container for holding one or more individual items of the identified category.

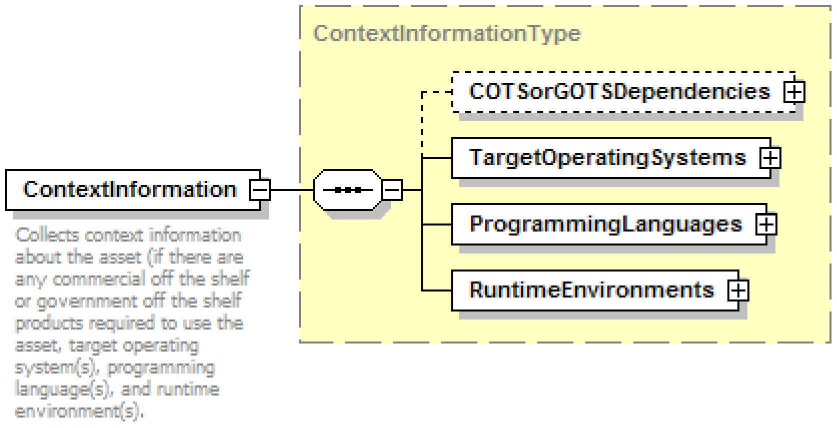

ContextInformationType
COTSorGOTSDependencies
TargetOperatingSystems
ProgrammingLanguages
RuntimeEnvironments
ContextInformation
Collects context information about the asset (if there are any commercial off the shelf or government off the shelf products required to use the asset, target operating system(s), programming language(s), and runtime environment(s).

Figure 14. ContextInformation Element

8. Artifacts Contained in the Asset

Assets contain artifacts. Two types of artifacts are currently in SHARE: document artifacts and code artifacts. This partition corresponds to the Asset Type information entered earlier in the process. The *ArtifactsContained* element (Figure 15) provides the top-level selection of one of these categories.

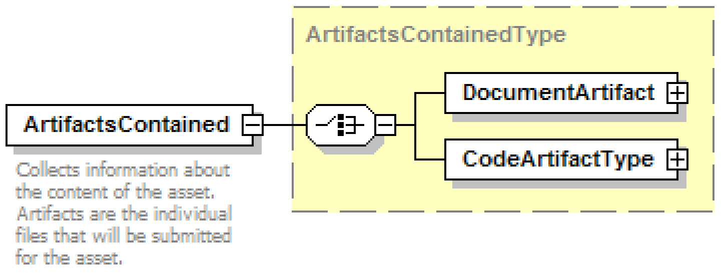

ArtifactsContainedType
ArtifactsContained
DocumentArtifact
CodeArtifactType
Collects information about the content of the asset. Artifacts are the individual files that will be submitted for the asset.

Figure 15. ArtifactsContained Element

For a document artifact, the *DocumentArtifact* structure (Figure 16) allows entry of artifacts falling into one or more of the six sub-categories: Requirements, Design/Architecture Documentation, Test Procedures, User Documentation, Training Documentation, or Test Records. These types are identified in the *ArtifactType* element of each *DocumentTypeArtifacts* element (Figure 17). The *Artifacts* element is a container for one or more individual *Artifact* elements identifying the artifact of the identified type contained in the asset being submitted.

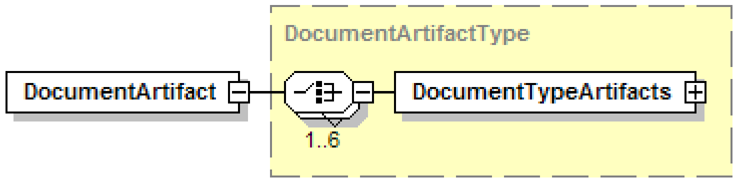


Figure 16. DocumentArtifact Element

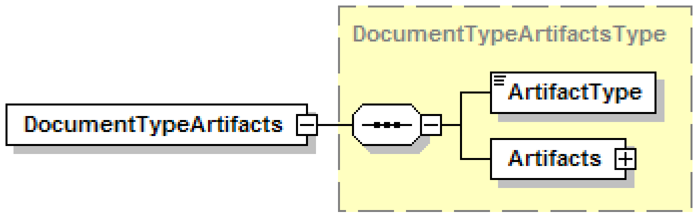


Figure 17. DocumentTypeArtifacts Element

For a code artifact, the *CodeArtifact* structure (Figure 18) allows entry of artifacts falling into one or more of the six sub-categories: Source Code, Compiled Libraries, Executable Programs, Data/Support Files, Build Scripts/Instructions, or Tools. These types are identified in the *ArtifactType* element of each *CodeTypeArtifacts* element (Figure 19). Again, the *Artifacts* element is a container for one or more individual *Artifact* elements identifying the artifact of the identified type contained in the asset being submitted.

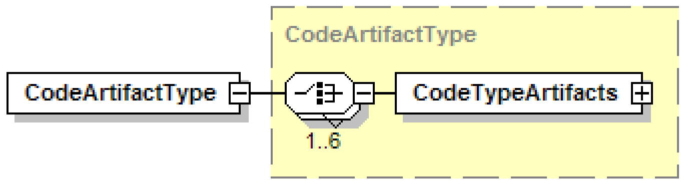


Figure 18. CodeArtifactType Element

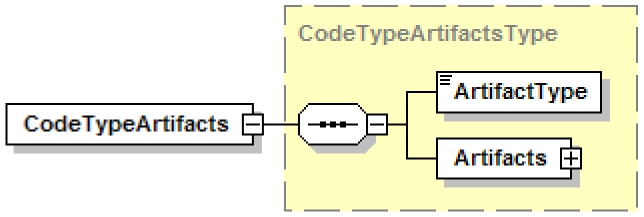
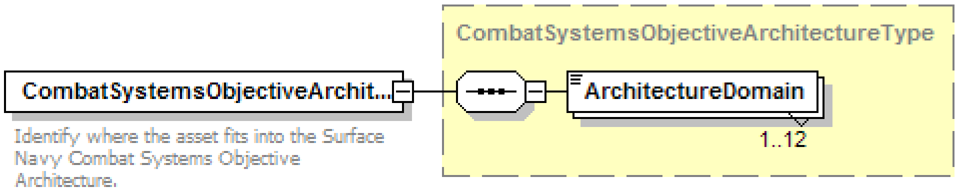


Figure 19. CodeTypeArtifacts Element

9. Architecture Domain

The Architecture Domain entry in the SHARE asset information form wizard allows the user to identify one or more domains of the Surface Navy Combat Systems Objective Architecture that apply to the asset being submitted. In the XML structure, the *CombatSystemsObjectiveArchitecture* (Figure 20) allows identification of one to 12 *ArchitectureDomain* elements, each having a value selected from the following: Computing Equipment, Infrastructure, Display, Sensor Management, Track Management, Command and Control (C2), Weapon Management, Vehicle Control, External Communications (EXCOMM), Common Support, Ship Control, and Other.

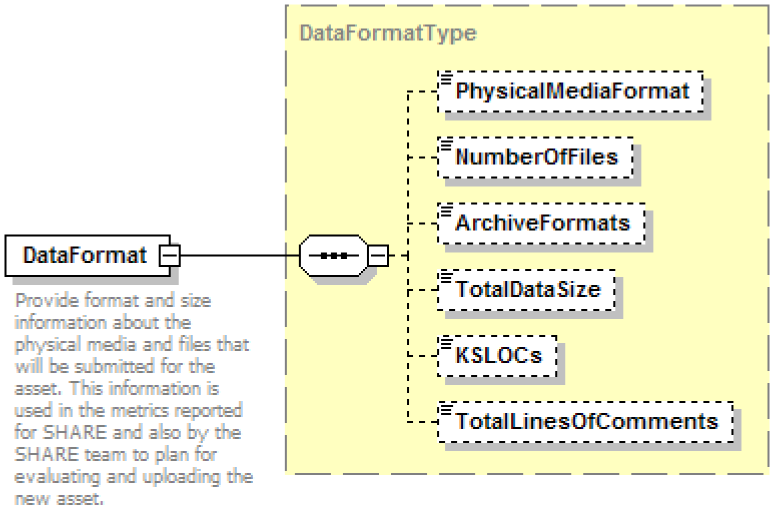


CombatSystemsObjectiveArchitectureType
CombatSystemsObjectiveArchit...
ArchitectureDomain
1..12
Identify where the asset fits into the Surface Navy Combat Systems Objective Architecture.

Figure 20. CombatSystemsObjectiveArchitecture Element

10. Data Format

The *DataFormat* element (Figure 21) describes the physical media format, number of files, archive formats, total data size, the number of source lines of code (in thousands), and the total lines of comments provided in the asset. Note that all this information is optional in the structure.



DataFormatType
PhysicalMediaFormat
NumberOfFiles
ArchiveFormats
TotalDataSize
KSLOCs
TotalLinesOfComments
DataFormat
Provide format and size information about the physical media and files that will be submitted for the asset. This information is used in the metrics reported for SHARE and also by the SHARE team to plan for evaluating and uploading the new asset.

Figure 21. DataFormat Element

11. Rights and Restrictions

The *RightsAndRestrictions* element (Figure 22) identifies any data rights marking contained with the asset being submitted and lists any other licenses that may be needed to use the asset. Each of the child elements (*DataRightsMarkings*, *CommercialSoftware*, *SpecialLicenses*, *OpenSourceSoftwareLicenses*, and *DataRightsAssertions*) are structured as choices between "yes" and "no" selections (*YesSelection* and *NoSelection* child elements) where the "yes" selection can contain a description of the pertinent information (as the value of the *YesSelection* element). *NoSelection* elements are empty.

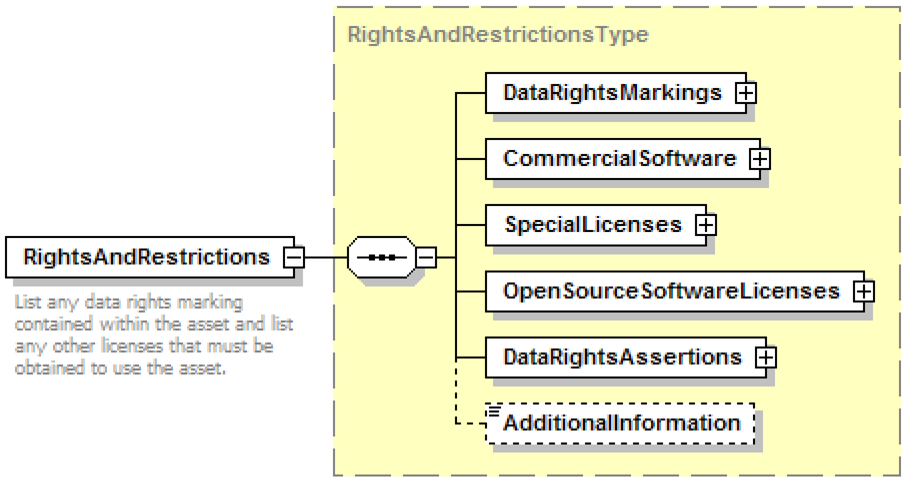

RightsAndRestrictionsType
DataRightsMarkings
CommercialSoftware
SpecialLicenses
OpenSourceSoftwareLicenses
DataRightsAssertions
AdditionalInformation
RightsAndRestrictions
List any data rights marking contained within the asset and list any other licenses that must be obtained to use the asset.

Figure 22. RightsAndRestrictions Element

C. Proposed Content Description: Separation of Assets and Artifacts

The most significant recommended change to the current SHARE approach to handling metadata is the level of application. It is our assertion that to enable the satisfaction of repository user needs, metadata must be applied at the artifact level rather than at the asset level, which is the current methodology for SHARE.

To be clear, we must provide our definition of these two concepts. The Navy Open Architecture (OA) program has adopted similar definitions for asset and artifact as those used in the Object Management Group (OMG) Reusable Asset Specification (RAS). In the RAS, artifacts are defined as "any work products from the software development lifecycle," and assets are a grouping of artifacts that "provide a solution to a problem for a given context" (Object Management Group, 2005, p. 7). Accordingly, the RAS describes an approach for packaging *artifacts* into an *asset*.

This is consistent with the current SHARE approach and remains consistent in the proposed metadata schema. However, the current SHARE approach is to package artifacts into assets at the convenience of the submitter and to enable the current retrieval process. We believe it is more useful to enable packaging of artifacts into assets based on users' needs. This means that the grouping of artifacts into an asset should have the capability of being user-defined. In order to enable this approach, the users must be able to discover the artifacts that are potentially of value to their particular context to solve a particular problem and then package those artifacts into an asset for retrieval.

Therefore, the proposed metadata schema includes separate definitions of structures for artifacts and assets. This does not preclude the pre-packaging of artifacts into assets for submission to the repository or for extraction to solve common problems. We envision the capability for users to discover a problem solution by either locating a prepackaged (reusable) asset or by building an asset from artifacts they believe will help solve their particular problem.

Splitting the metadata into two schemas, one for assets and another for artifacts, also enables a clearer distinction about the data that needs to be collected for each. For example, the current SHARE metadata collects data on the types of artifacts included in the asset, such as whether they are documents or code. Then, it separately asks for thousands of lines of code (KSLOC) for the asset. This would more likely be tied to particular artifacts that are of the type "code" in the asset. By separating the asset and artifact schemas, we can better distinguish the data that is necessary for an asset from the data that is necessary for an artifact, and we will be able to manipulate the data more appropriately with tools that implement the search.

Collecting metadata information for each artifact may seem like a daunting task when compared to the current method. However, it is highly likely that a good portion of the metadata that applies to one artifact also applies to the remaining artifacts in a group of artifacts being submitted. The submission tool could be constructed in such a way to minimize duplicative entries of data by prompting users to verify that the information being entered applies to all of the artifacts in a group. This would minimize the individual entries required in the submission and metadata collection process. It is also possible to create tools that automate much of the metadata collection from the artifacts themselves. Other organizations are conducting research and development to auto-generate metadata from the source products. This is a critical capability in making legacy content available for search and discovery. Adoption of structured metadata makes autogeneration feasible, although certainly nontrivial. This is a recommended area for future research and development in the SHARE program.

1. Artifacts Schema

The artifacts schema is provided in Appendix B. The following sections describe the schema design and details.

Schema overview

The artifacts schema is designed to be flexible in its implementation. All the elements, types and attributes in the schema are defined globally so they can be reused in other schemas that developers may create for working with artifact information. The root element, *Artifacts*, is simply a container for any number of artifacts contained in a single instance of the schema, as shown in Figure 23. Repository managers and tool designers can decide if they wish to keep a separate XML file describing each artifact or if they prefer to group multiple artifact descriptions into a single XML file.

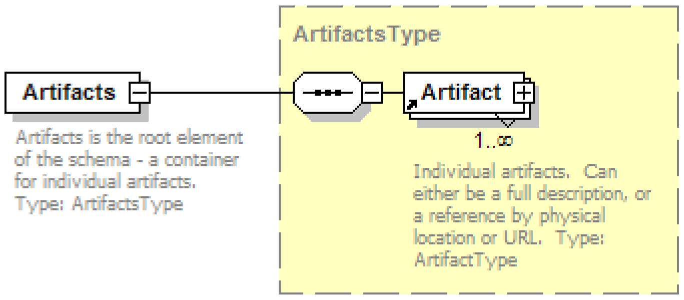

ArtifactsType
Artifacts
Artifacts is the root element of the schema - a container for individual artifacts. Type: ArtifactsType
Artifact
1..∞
Individual artifacts. Can either be a full description, or a reference by physical location or URL. Type: ArtifactType

Figure 23. Artifacts Element

The individual descriptions of each of the artifacts are also designed to be flexible. A specific artifact can be incorporated into the file in one of three ways. The first is by providing the full artifact description. This full description represents the heart of the metadata development effort and should be considered the preferred method for representing an artifact. However, if the full description is not available, or if the information required is provided in some other location, the schema allows the inclusion of the artifact representation by reference, either to a physical location or by URL. This is shown in Figure 24.

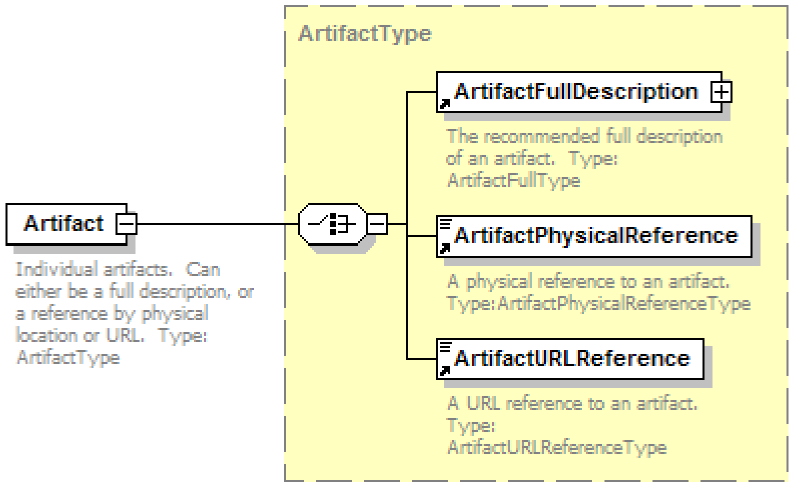

ArtifactType
Artifact
Individual artifacts. Can either be a full description, or a reference by physical location or URL. Type: ArtifactType
ArtifactFullDescription
The recommended full description of an artifact. Type: ArtifactFullType
ArtifactPhysicalReference
A physical reference to an artifact. Type:ArtifactPhysicalReferenceType
ArtifactURLReference
A URL reference to an artifact. Type: ArtifactURLReferenceType

Figure 24. Artifact Element

Artifact Full Description

The full description of each artifact, contained in the element *ArtifactFullDescription*, is composed of eight subelements as depicted in Figure 25. Each of these subelements is discussed in detail in the following sections.

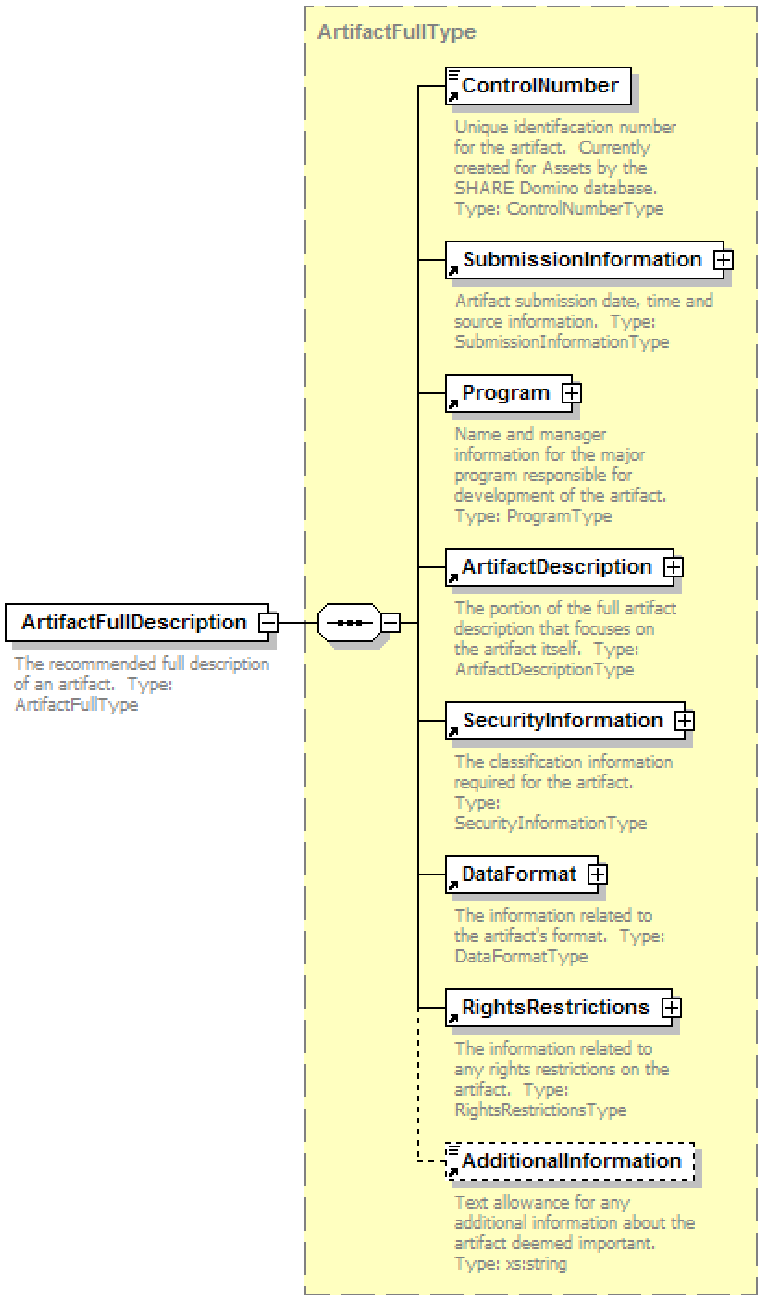

ArtifactFullType
ControlNumber
Unique identifacation number for the artifact. Currently created for Assets by the SHARE Domino database. Type: ControlNumberType
SubmissionInformation
Artifact submission date, time and source information. Type: SubmissionInformationType
Program
Name and manager information for the major program responsible for development of the artifact. Type: ProgramType
ArtifactDescription
The portion of the full artifact description that focuses on the artifact itself. Type: ArtifactDescriptionType
ArtifactFullDescription
The recommended full description of an artifact. Type: ArtifactFullType
SecurityInformation
The classification information required for the artifact. Type: SecurityInformationType
DataFormat
The information related to the artifact's format. Type: DataFormatType
RightsRestrictions
The information related to any rights restrictions on the artifact. Type: RightsRestrictionsType
AdditionalInformation
Text allowance for any additional information about the artifact deemed important. Type: xs:string

Figure 25. ArtifactFullDescription Element

ControlNumber

The *ControlNumber* element contains a unique identifier for the artifact. Currently, this number is a 32-digit hexadecimal number automatically generated by the SHARE Domino Database. It is worth noting that if the artifacts are meant to be shared as part of an enterprise repository design, the origin of this number may need to be reconsidered in order to avoid any chance of duplication.

SubmissionInformation

The *SubmissionInformation* element contains information about the date, time and source of the artifact as shown in Figure 26.

For the source, information is collected for both the individual submitting the artifact as well as the organization responsible for its creation as shown in Figure 27. In the current SHARE submission process, the point of contact information about the individual submitting an asset is pulled automatically from the database for the person who is logged into the repository during the submission. Point of contact information is used in several locations throughout the schema. While the individual elements that make up this global element are depicted here, they will be assumed for all elements with *Type: POCInfo* in later diagrams and not shown in the decomposed state for brevity.

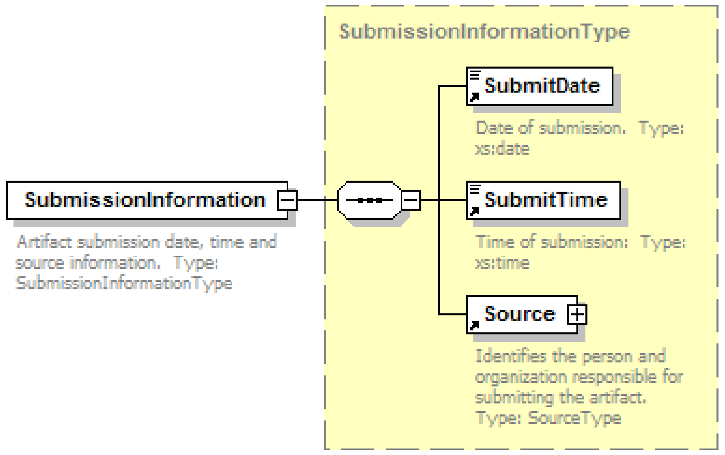

SubmissionInformationType
SubmitDate
Date of submission. Type: xs:date
SubmissionInformation
Artifact submission date, time and source information. Type: SubmissionInformationType
SubmitTime
Time of submission: Type: xs:time
Source
Identifies the person and organization responsible for submitting the artifact. Type: SourceType

Figure 26. SubmissionInformation Element

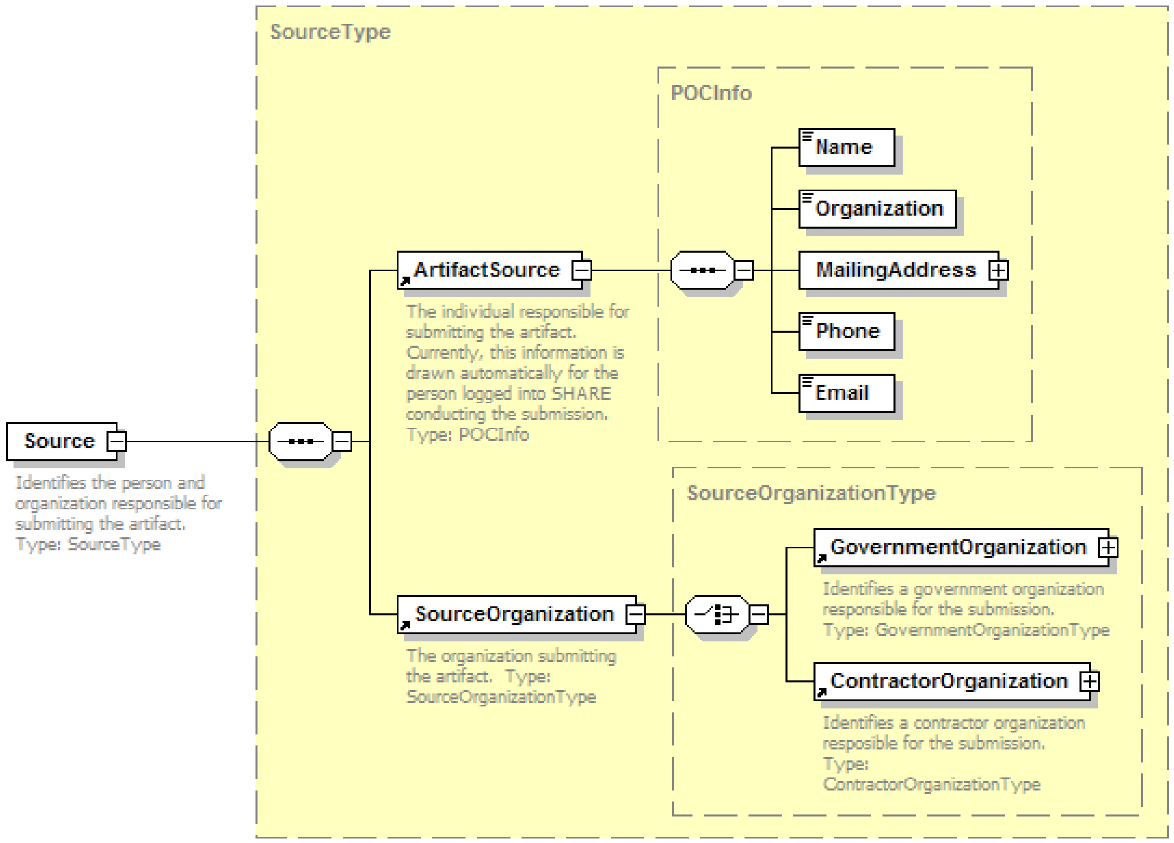

SourceType
POCInfo
Name
Organization
MailingAddress
Phone
Email
ArtifactSource
The individual responsible for submitting the artifact. Currently, this information is drawn automatically for the person logged into SHARE conducting the submission. Type: POCInfo
Source
Identifies the person and organization responsible for submitting the artifact. Type: SourceType
SourceOrganizationType
GovernmentOrganization
Identifies a government organization responsible for the submission. Type: GovernmentOrganizationType
SourceOrganization
The organization submitting the artifact. Type: SourceOrganizationType
ContractorOrganization
Identifies a contractor organization resposible for the submission. Type: ContractorOrganizationType

Figure 27. Source Element

The *SourceOrganization* can either be a government or contractor agency. Different metadata is required in each case. For a government organization, the information required is the organization name and a list of any relevant patents for the artifact. This is shown in Figure 28.

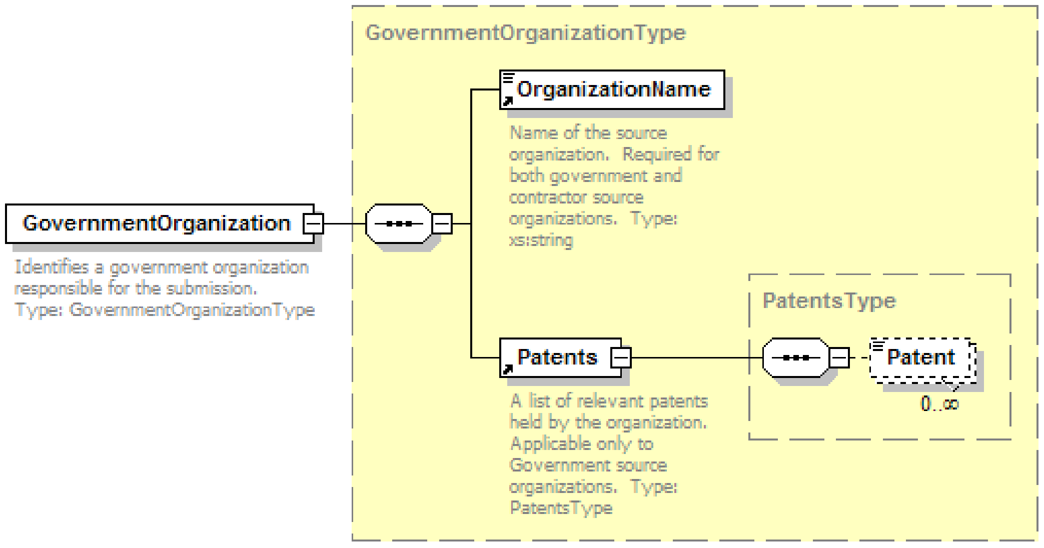

GovernmentOrganizationType
OrganizationName
Name of the source organization. Required for both government and contractor source organizations. Type: xs:string
GovernmentOrganization
Identifies a government organization responsible for the submission. Type: GovernmentOrganizationType
PatentsType
Patents
Patent
0..∞
A list of relevant patents held by the organization. Applicable only to Government source organizations. Type: PatentsType

Figure 28. GovernmentOrganization Element

The *ContractorOrganization* element is shown in Figure 29. As in the *GovernmentOrganization*, required elements include the organization name and type. However, the contract number, delivery vehicle, and Contracting Officers Representative (COR) applicable to the artifact are additional required elements. The value options for the delivery vehicle include DD250, CDRL or Don’t Know, as is indicated in the current SHARE submission process.

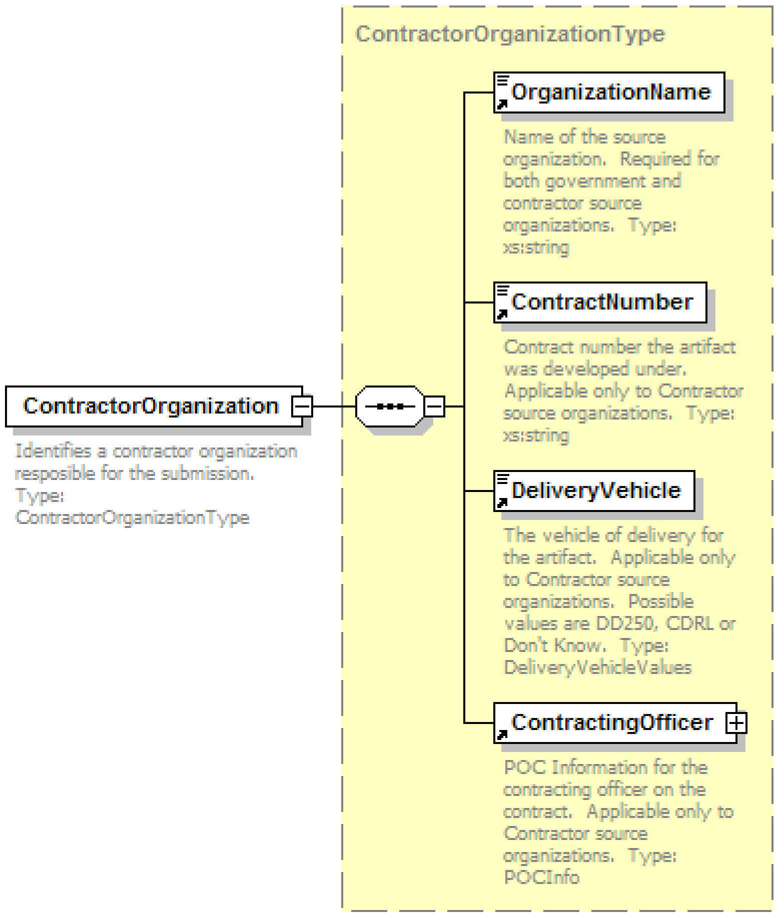

ContractorOrganizationType
OrganizationName
Name of the source organization. Required for both government and contractor source organizations. Type: xs:string
ContractNumber
Contract number the artifact was developed under. Applicable only to Contractor source organizations. Type: xs:string
ContractorOrganization
Identifies a contractor organization resposible for the submission. Type: ContractorOrganizationType
DeliveryVehicle
The vehicle of delivery for the artifact. Applicable only to Contractor source organizations. Possible values are DD250, CDRL or Don't Know. Type: DeliveryVehicleValues
ContractingOfficer
POC Information for the contracting officer on the contract. Applicable only to Contractor source organizations. Type: POCInfo

Figure 29. ContractorOrganization Element

Program

The *Program* element in the *ArtifactFullType* structure captures the name and manager of the major Program of Record (POR) responsible for the creation of the artifact and is shown in Figure 30. The artifact may be a product of one of the programs already contained in SHARE, or the program may be new to SHARE. For this reason, we have incorporated a choice between the two types of names. If the name is an ExistingProgramName, its value is limited to the existing list of programs in SHARE. Currently, this list includes AEGIS, DDG 1000, SSDS, LCS, NSWCDD HSI, SIAP, SQQ-89, TSTS, ASW, CEP WASP, GeDear, and BFTT. If the program does not already belong to the enumeration list, the new name is entered as a

NewProgramName. This distinction is necessary to allow the submission tool to handle the two names differently. If the name belongs to the existing list, the program manager POC information can be automatically pulled from the database. Otherwise, the tool will need to prompt users to input the information.[9]

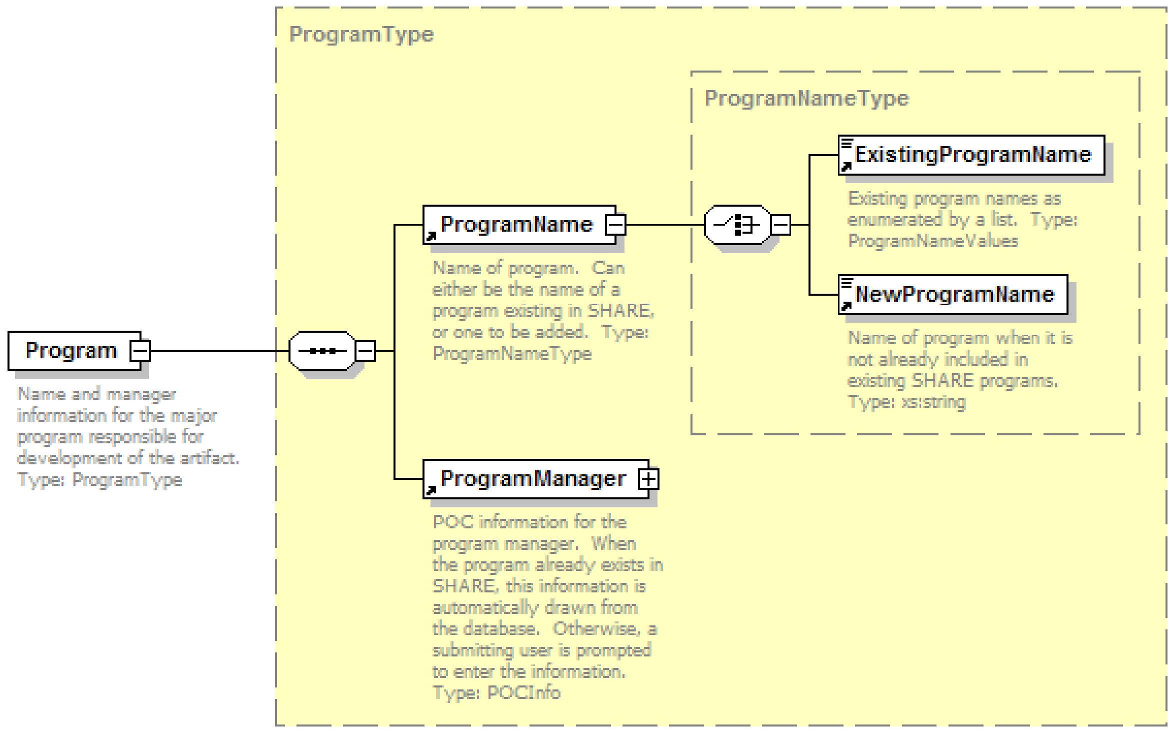

ProgramType
ProgramNameType
ExistingProgramName
Existing program names as enumerated by a list. Type: ProgramNameValues
ProgramName
Name of program. Can either be the name of a program existing in SHARE, or one to be added. Type: ProgramNameType
NewProgramName
Name of program when it is not already included in existing SHARE programs. Type: xs:string
Program
Name and manager information for the major program responsible for development of the artifact. Type: ProgramType
ProgramManager
POC information for the program manager. When the program already exists in SHARE, this information is automatically drawn from the database. Otherwise, a submitting user is prompted to enter the information. Type: POCInfo

Figure 30. Program Element

ArtifactDescription

The *ArtifactDescription* element is the heart of the *FullArtifactDescription*. The information that will influence the decision of whether to retrieve an item is most likely to be found in this element since the artifacts themselves are described therein. The first level of decomposition of the *ArtifactDescription* is provided in Figure 31.

[9] Software can then modify this governing schema to add the new program name to the list of previously identified programs for validating subsequent XML descriptions of artifacts.

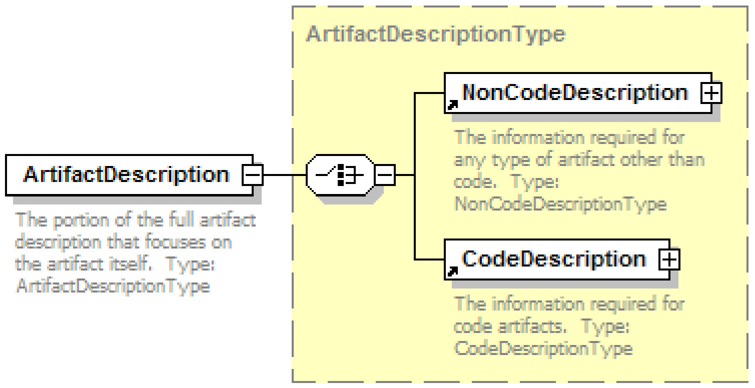

ArtifactDescriptionType
NonCodeDescription
The information required for any type of artifact other than code. Type: NonCodeDescriptionType
ArtifactDescription
The portion of the full artifact description that focuses on the artifact itself. Type: ArtifactDescriptionType
CodeDescription
The information required for code artifacts. Type: CodeDescriptionType

Figure 31. ArtifactDescription Element

The information necessary to describe the artifacts differs depending on whether the artifact is software code or some other type. Therefore, the schema allows a choice between the two types of artifact descriptions. The *NonCodeDescription* element applies to any artifact not considered software code. The group of elements contained therein is also required for artifacts that fall under the *CodeDescription* element category, but additional elements are required for code artifacts. To facilitate the use of elements in both cases, we built a group of elements called *ItemDescriptionGroup* as shown in Figure 32. Each of the elements contained in the *ItemDescriptionGroup*, and therefore as part of the *NonCodeDescription* element, is described in further detail below.

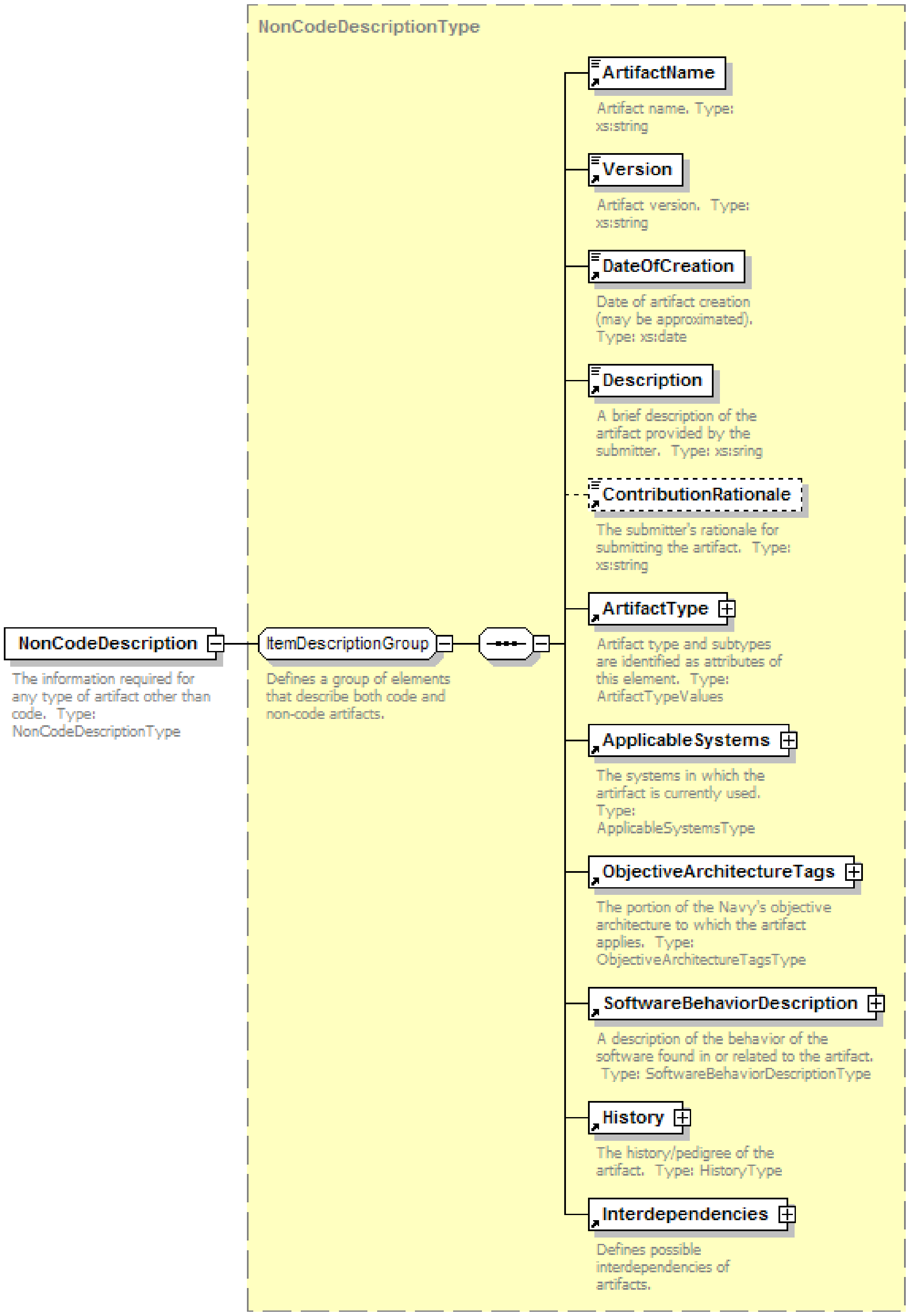

NonCodeDescriptionType
ArtifactName
Artifact name. Type: xs:string
Version
Artifact version. Type: xs:string
DateOfCreation
Date of artifact creation (may be approximated). Type: xs:date
Description
A brief description of the artifact provided by the submitter. Type: xs:sring
ContributionRationale
The submitter's rationale for submitting the artifact. Type: xs:string
ArtifactType
Artifact type and subtypes are identified as attributes of this element. Type: ArtifactTypeValues
NonCodeDescription
The information required for any type of artifact other than code. Type: NonCodeDescriptionType
ItemDescriptionGroup
Defines a group of elements that describe both code and non-code artifacts.
ApplicableSystems
The systems in which the artirfact is currently used. Type: ApplicableSystemsType
ObjectiveArchitectureTags
The portion of the Navy's objective architecture to which the artifact applies. Type: ObjectiveArchitectureTagsType
SoftwareBehaviorDescription
A description of the behavior of the software found in or related to the artifact. Type: SoftwareBehaviorDescriptionType
History
The history/pedigree of the artifact. Type: HistoryType
Interdependencies
Defines possible interdependencies of artifacts.

Figure 32. NonCodeDescription Element

The first three elements in the group—*ArtifactName*, *Version*, and *DateofCreation*—are relatively straightforward. The artifact name is assigned by the artifact submitter usually according to the naming convention followed by the POR responsible for its creation. The version assignment should follow the configuration management process of the POR. The date of creation may be approximated if the actual date is not known.

The *Description* and *ContributionRationale* elements are subjective. The artifact submitter will supply a free text entry for each element, with the contribution rationale only as an optional entry.

The *ArtifactType* element refers to the classification of artifact type according to the artifact taxonomy incorporated into the artifacts-lifecycle ontology described in section V.B. The assignment of a type and subtype attribute identifies the artifact's place in the ontology as shown in Figure 33.

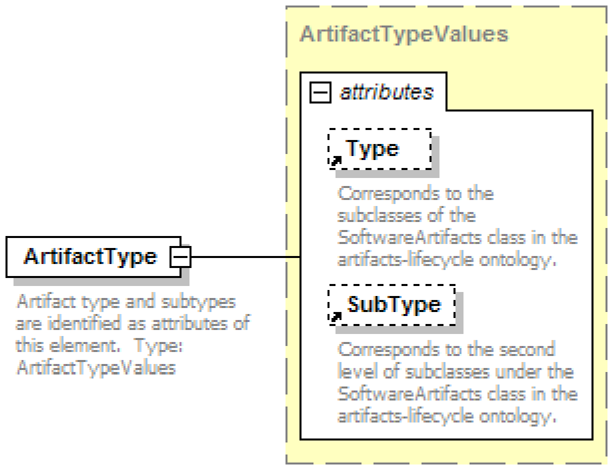

ArtifactTypeValues
attributes
Type
Corresponds to the subclasses of the SoftwareArtifacts class in the artifacts-lifecycle ontology.
ArtifactType
Artifact type and subtypes are identified as attributes of this element. Type: ArtifactTypeValues
SubType
Corresponds to the second level of subclasses under the SoftwareArtifacts class in the artifacts-lifecycle ontology.

Figure 33. ArtifactType Element

A detailed description of the ontology and its intended use are provided in section V. However, the taxonomy for artifact Types and SubTypes is shown in Table 1. The repository tool should verify the values that are assigned for the Type and SubType attributes of the ArtifactType element against this taxonomy, which is included in the artifact-lifecycle ontology.

Table 1. Artifact Type Taxonomy

Type	SubType
Requirements Artifacts	Requirements Database
	Requirements Specification
Design Artifacts	Algorithm
	Data Model
	Design Document
	Design Model
	Pattern
Architecture Artifacts	Architecture Document
	Architecture Model
Code Artifacts	Compiled Library
	Executable Program
	Source Code
Simulation Artifacts	Simulation Model
	Simulator
Interface Artifacts	Application Programming Interface
	Interface Design Document
	Interface Design Specification
	Interface Requirements Specification
Test Artifacts	Test Plan
	Test Procedure
	Test Result
	Test Script
	Test Source Data File
	Test Tool
	Test Truth Data
User Artifacts	Build Instruction
	Build Script
	Training Documentation
	User Documentation
Other Artifacts	White Paper

The *ApplicableSystems* element is a list of the major systems and their subsystems for which the artifact has previously been used and is shown in Figure 34. As we will discuss in the ontology discussion, the ideal repository framework will incorporate system-subsystem ontologies for each system contained in the repository against which the ApplicableSystems sub-elements can be verified. This will allow further use of this field in defining context for the artifacts. Until that level of detail is implemented, knowledge of the applicable System and Subsystem pairs is still valuable. Ideally, the subsystem identified should reflect the lowest level of granularity covered by the artifact. Since an artifact may be relevant to an entire system or systems, the *Subsystem* element is captured as an optional element in the schema.

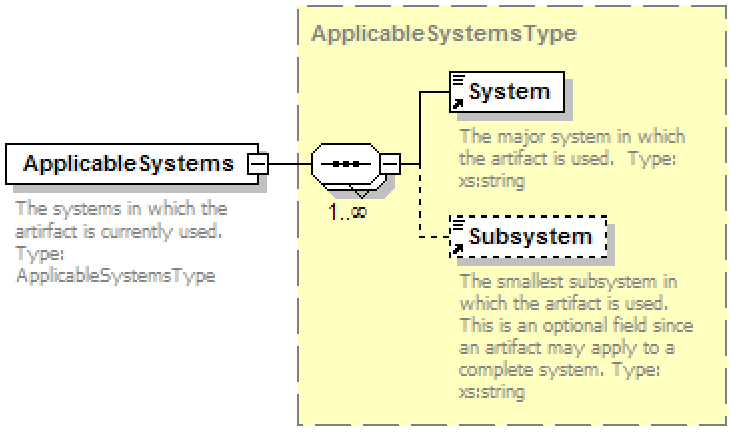

ApplicableSystemsType
System
The major system in which the artifact is used. Type: xs:string
ApplicableSystems
The systems in which the artifact is currently used. Type: ApplicableSystemsType
1..∞
Subsystem
The smallest subsystem in which the artifact is used. This is an optional field since an artifact may apply to a complete system. Type: xs:string

Figure 34. ApplicableSystems Element

The *ObjectiveArchitectureTags* element is displayed in Figure 35. This element is similar to the *ArtifactType* element in that the allowable values for the subelements, *DomainTags* and *SubDomainTags*, should be verified against a separate ontology. This ontology is based on the Surface Combat System Top Level Objective Architecture built under the Navy OA program. The *DomainTags* and *SubDomainTags* correspond to the two levels of decomposition represented by the Objective Architecture, thus identifying the relationship of the artifact to the architecture. Since each artifact may apply to multiple domains or sub-domains, the elements are constructed as containers capable of one to many individual entries. The objective architecture ontology and its utility are discussed further in section V.C.

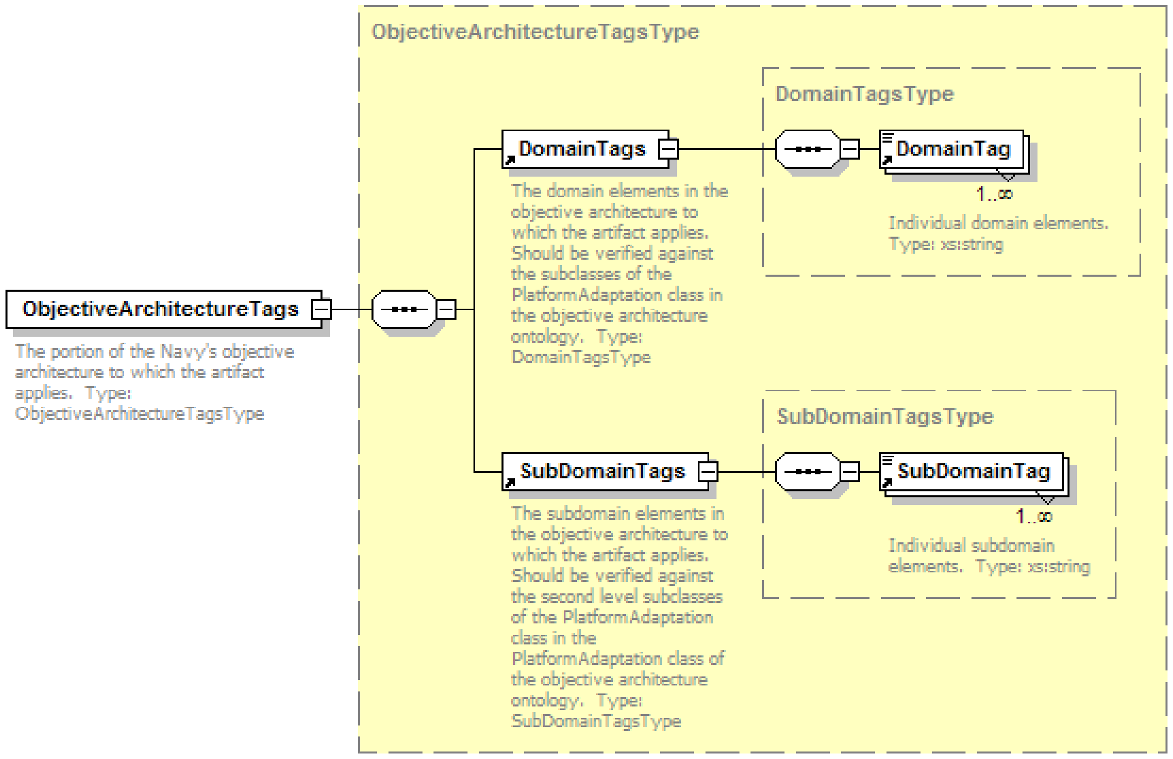

ObjectiveArchitectureTagsType
DomainTagsType
DomainTags
DomainTag
1..∞
The domain elements in the objective architecture to which the artifact applies. Should be verified against the subclasses of the PlatformAdaptation class in the objective architecture ontology. Type: DomainTagsType
Individual domain elements. Type: xs:string
ObjectiveArchitectureTags
The portion of the Navy's objective architecture to which the artifact applies. Type: ObjectiveArchitectureTagsType
SubDomainTagsType
SubDomainTags
SubDomainTag
1..∞
Individual subdomain elements. Type: xs:string
The subdomain elements in the objective architecture to which the artifact applies. Should be verified against the second level subclasses of the PlatformAdaptation class in the PlatformAdaptation class of the objective architecture ontology. Type: SubDomainTagsType

Figure 35. ObjectiveArchitectureTags Element

The *SoftwareBehaviorDescription* element captures behavioral information about the artifact as shown in Figure 36. This behavior is ideally captured in two ways. First, the functionality of the software related to the artifact is identified by a list of functions selected from the Common System Function List. We have converted the Navy's Common Systems Function List (CSFL) into an ontology expressed in the Web Ontology Language (OWL). Acceptable entries for the *CommonSystemFunction* element should be validated against this ontology. Second, the interface information is captured as a Web Service Description Language (WSDL) document. The CSFL ontology and the WSDL descriptions of the artifacts are discussed in more detail in section IV.

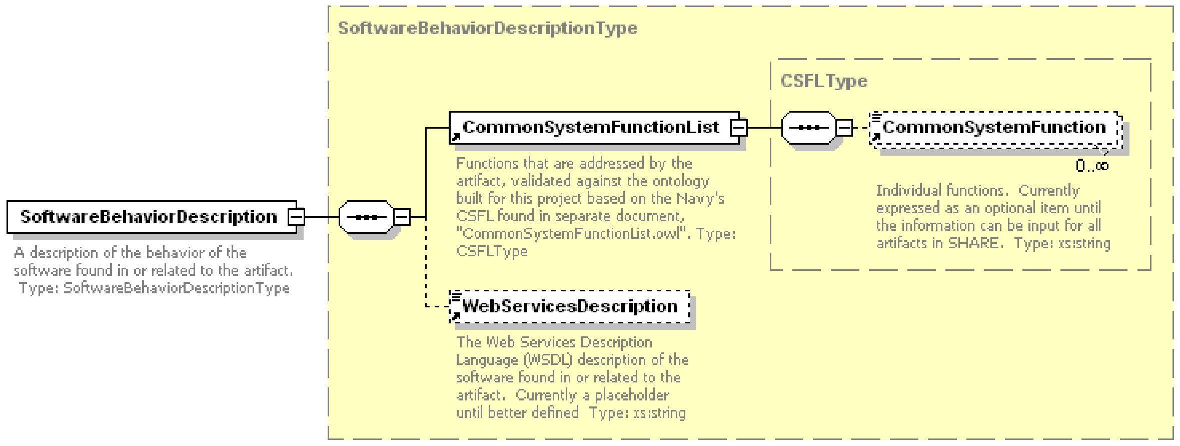

SoftwareBehaviorDescriptionType
CSFLType
SoftwareBehaviorDescription
A description of the behavior of the software found in or related to the artifact. Type: SoftwareBehaviorDescriptionType
CommonSystemFunctionList
Functions that are addressed by the artifact, validated against the ontology built for this project based on the Navy's CSFL found in separate document, "CommonSystemFunctionList.owl". Type: CSFLType
CommonSystemFunction
0..∞
Individual functions. Currently expressed as an optional item until the information can be input for all artifacts in SHARE. Type: xs:string
WebServicesDescription
The Web Services Description Language (WSDL) description of the software found in or related to the artifact. Currently a placeholder until better defined Type: xs:string

Figure 36. SoftwareBehaviorDescription Element

The *History* element captures information about the status and background of the artifact and is depicted in Figure 37. The *DevelopmentStatus* indicates whether an artifact is complete, with restricted allowable values of InDevelopment or DevelopmentComplete. *PlannedUpdates* and *MaturityDescription* are free text notes that can be added by the artifact submitter.

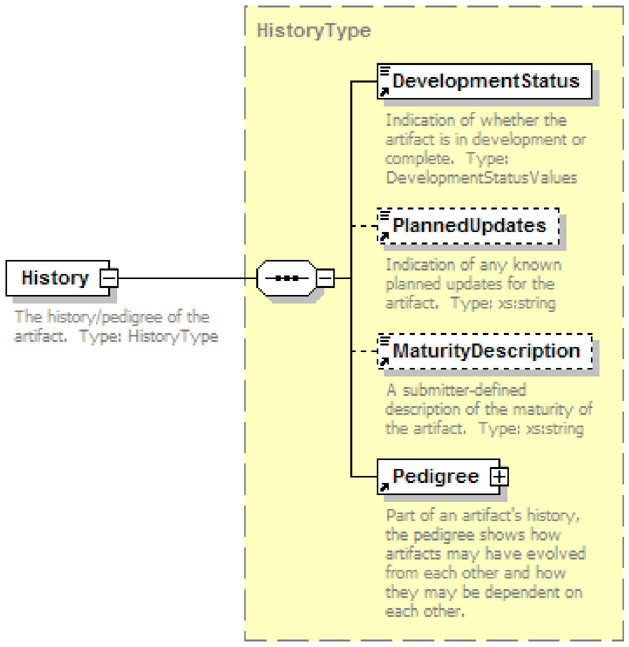

HistoryType
History
The history/pedigree of the artifact. Type: HistoryType
DevelopmentStatus
Indication of whether the artifact is in development or complete. Type: DevelopmentStatusValues
PlannedUpdates
Indication of any known planned updates for the artifact. Type: xs:string
MaturityDescription
A submitter-defined description of the maturity of the artifact. Type: xs:string
Pedigree
Part of an artifact's history, the pedigree shows how artifacts may have evolved from each other and how they may be dependent on each other.

Figure 37. History Element

The *Pedigree* subelement of *History* is provided in Figure 38. In the current SHARE repository, asset submitters are asked to relate the asset being submitted to existing items in the repository according to one or more of the relationships listed as subelements in the *History* element. The current SHARE definitions of these relationships are captured in Table 2.

With the incorporation of the ontologies into the repository framework, we believe that many of these relationships could be derived automatically. For example, an artifact of the same name, type, system/subsystem mapping, and a later version number could be assumed to be a newer version of an existing artifact. Similar logic could be applied to derive several of the other relationships captured here. Rule-dependent, automatic derivation of these relationships is highly desirable since manual assignment is likely to be sporadic and inconsistent.

Further work is required to flesh out the intended specific meanings of several relationships as currently defined and to determine the appropriate rules that would enable automatic relationship assignment. Until this work is completed, we decided to include the relationships in their current form but to apply them at the more meaningful artifact level. Additionally, we moved the Dependent Upon, Needed By, and Interfaces With relationships under the *Interdependencies* element discussed in the following paragraph.

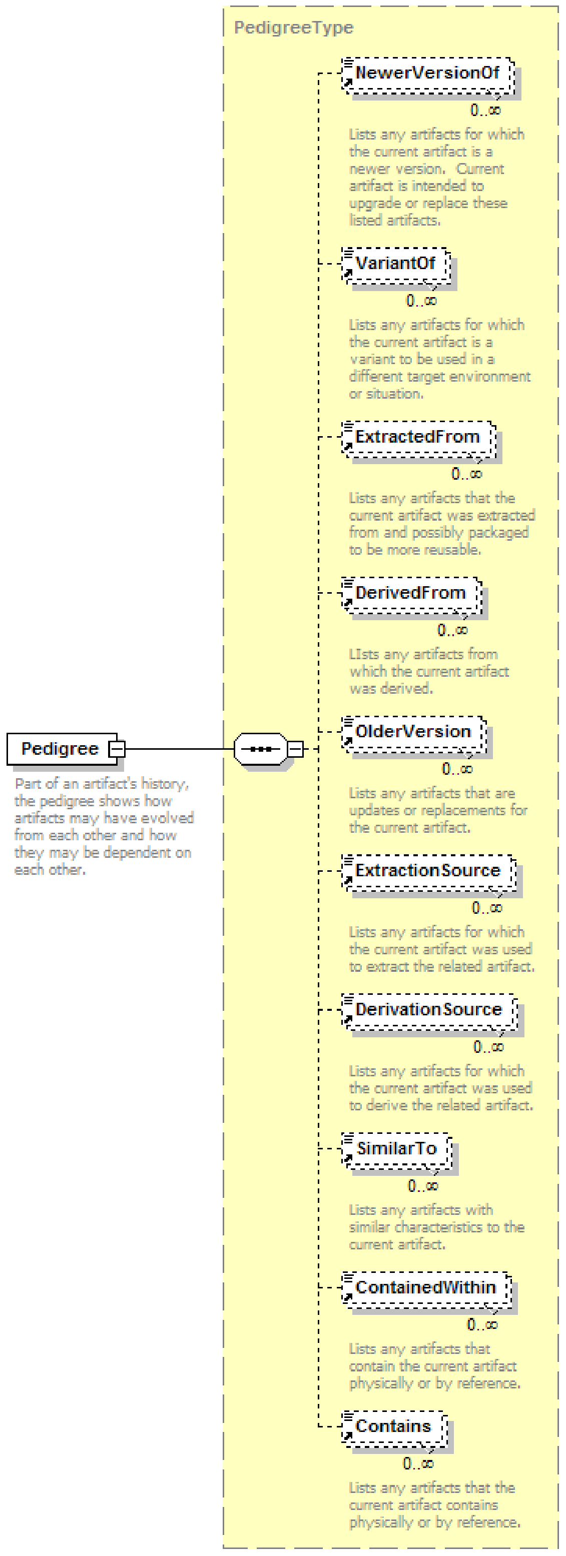

PedigreeType
Pedigree
Part of an artifact's history, the pedigree shows how artifacts may have evolved from each other and how they may be dependent on each other.
NewerVersionOf
0..∞
Lists any artifacts for which the current artifact is a newer version. Current artifact is intended to upgrade or replace these listed artifacts.
VariantOf
0..∞
Lists any artifacts for which the current artifact is a variant to be used in a different target environment or situation.
ExtractedFrom
0..∞
Lists any artifacts that the current artifact was extracted from and possibly packaged to be more reusable.
DerivedFrom
0..∞
LIsts any artifacts from which the current artifact was derived.
OlderVersion
0..∞
Lists any artifacts that are updates or replacements for the current artifact.
ExtractionSource
0..∞
Lists any artifacts for which the current artifact was used to extract the related artifact.
DerivationSource
0..∞
Lists any artifacts for which the current artifact was used to derive the related artifact.
SimilarTo
0..∞
Lists any artifacts with similar characteristics to the the current artifact.
ContainedWithin
0..∞
Lists any artifacts that contain the current artifact physically or by reference.
Contains
0..∞
Lists any artifacts that the current artifact contains physically or by reference.

Figure 38. *Pedigree* Element

Table 2. SHARE Asset Relationships (current)

Relationship	Description
Was this asset created from other assets in SHARE?	
Newer Version	This indicates that the current asset is an earlier version of the related asset. It is intended to upgrade or replace the related asset, not be a variant of it.
Variant Of	This indicates that the current asset is a variant of the related asset to be used in a different target environment or situation.
Extracted From	This indicates that the current asset was extracted from another asset and was possibly repackaged to be more reusable. The extracted asset will not be merged back into the asset it was extracted from.
Derived From	This indicates that the current asset was originally based on the related asset but is sufficiently different to consider it a different asset and not a version or variant.
Does this asset work in conjunction with other assets in SHARE?	
Dependent Upon	This indicates that the current asset references or relies on the services or artifacts of the related asset to provide the desired/required capability.
Needed By	This indicates that the current asset is referenced or used by the related asset to provide the desired/required capability.
Interfaces With	This indicates that the current asset communicates with the related assets. The related assets may be needed for operation but they are not needed to complete the requirements.
Were other assets in SHARE created from this asset?	
Older Version	This indicates that the current asset is updated or replaced by the related asset.
Extraction Source	This indicates that the current asset was used to extract the related asset. The extracted asset will not be merged back into the asset it was extracted from.
Derivation Source	This indicates that the related asset was originally based on the current asset but is sufficiently different to consider it a different asset and not a version or variant.
Is this asset related in other ways to other assets in SHARE?	
Similar To	This indicates that the other asset has characteristics similar to the current asset.
Contained Within	This indicates that the current asset is part of a larger asset that is also stored in SHARE. This containment may be physical or by reference.
Contains	This indicates that the current asset contains the related asset. This containment may be physical or by reference.

The *Interdependencies* element of the *NonCodeDescription* structure is shown in Figure 39. The first two subelements, *DependentUpon* and *NeededBy*, are carried over from the SHARE asset relationships described in Table 2. We consider these items to be somewhat vague and recommend that further work be done to refine the intended meaning of the relationships. If the criteria for classification in one of these two categories are better defined, the ontologies may provide the necessary links to automatically assign these relationships between artifacts. The *COTS_GOTSDependencies* element captures any dependencies the artifact may have on items that may be outside configuration control of the asset submitter. Finally, the InterfacesWith element indicates any artifacts with which the artifact communicates. If the repository framework is eventually populated with ontologies that represent each system's architecture as described above, this item may be automatically populated as well.

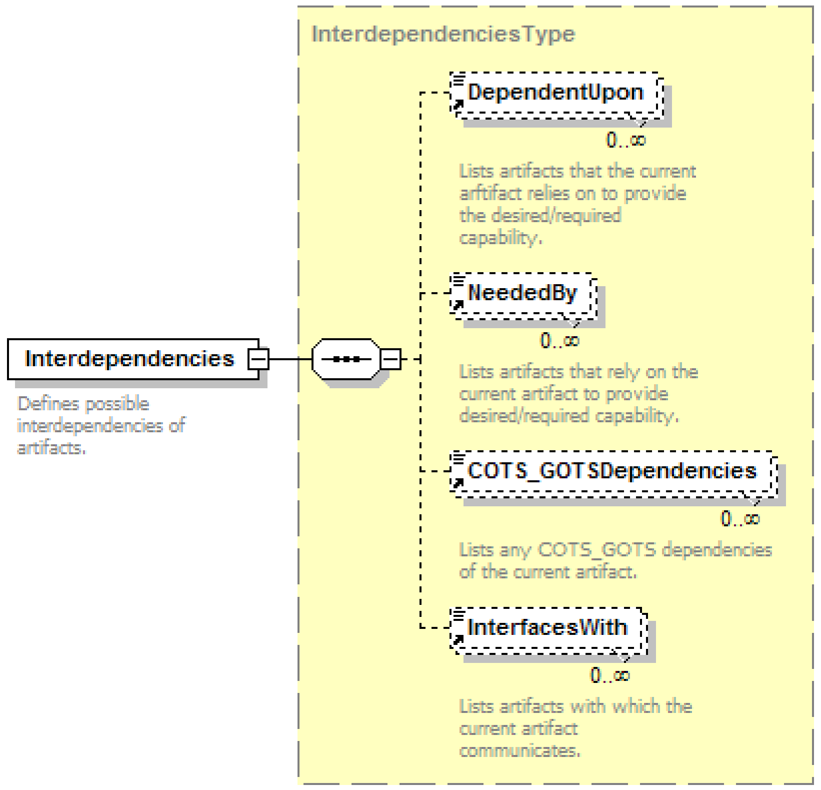

InterdependenciesType
Interdependencies
Defines possible interdependencies of artifacts.
DependentUpon
0..∞
Lists artifacts that the current arftifact relies on to provide the desired/required capability.
NeededBy
0..∞
Lists artifacts that rely on the current artifact to provide desired/required capability.
COTS_GOTSDependencies
0..∞
Lists any COTS_GOTS dependencies of the current artifact.
InterfacesWith
0..∞
Lists artifacts with which the current artifact communicates.

Figure 39. Interdependencies Element

All the elements described above as part of the *ItemDescriptionGroup* are included in the *CodeDescription* element as well as the *NonCodeDescription*, as shown in Figure 40. However, there are additional fields required for code items. They include information about the target operating system, programming languages used, the intended runtime environment and size information, such as KSLOC (thousands of source lines of code) and total lines of comments.

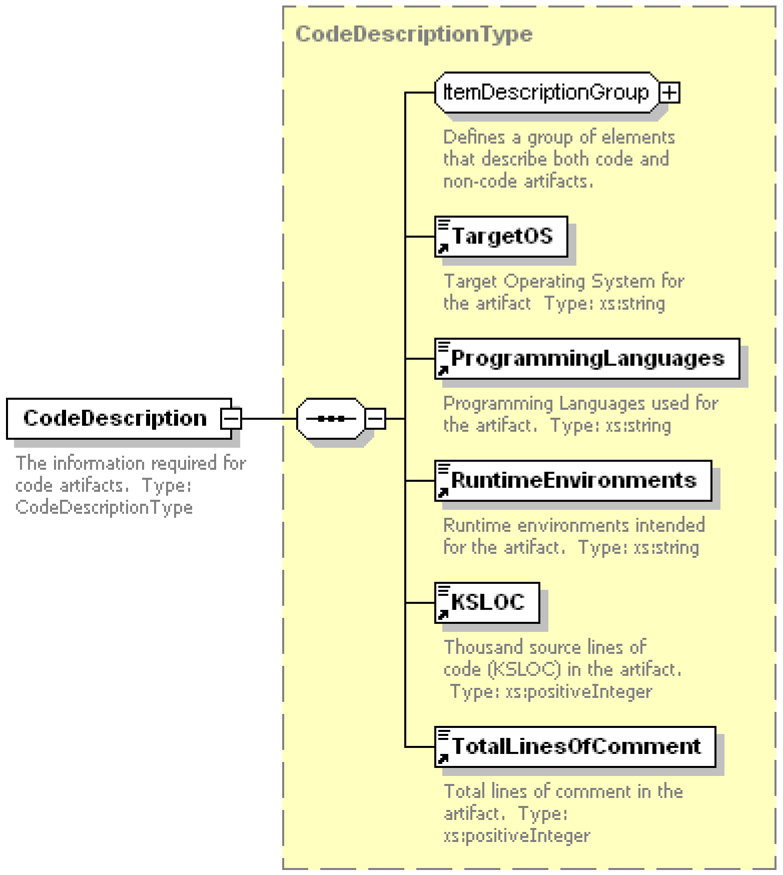

CodeDescriptionType
ItemDescriptionGroup
Defines a group of elements that describe both code and non-code artifacts.
TargetOS
Target Operating System for the artifact Type: xs:string
ProgrammingLanguages
Programming Languages used for the artifact. Type: xs:string
CodeDescription
The information required for code artifacts. Type: CodeDescriptionType
RuntimeEnvironments
Runtime environments intended for the artifact. Type: xs:string
KSLOC
Thousand source lines of code (KSLOC) in the artifact. Type: xs:positiveInteger
TotalLinesOfComment
Total lines of comment in the artifact. Type: xs:positiveInteger

Figure 40. CodeDescription Element

SecurityInformation

The next element included in the full artifact description is *SecurityInformation*. Different information is required for classified artifacts than unclassified, so the element is divided into two choices, as shown in Figure 41.

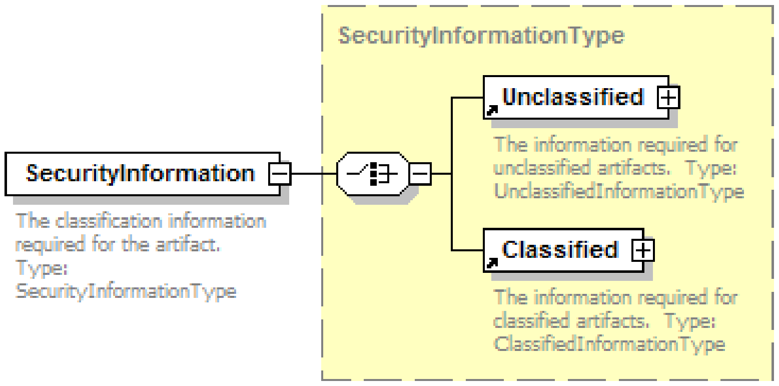

SecurityInformationType
Unclassified
The information required for unclassified artifacts. Type: UnclassifiedInformationType
SecurityInformation
The classification information required for the artifact. Type: SecurityInformationType
Classified
The information required for classified artifacts. Type: ClassifiedInformationType

Figure 41. SecurityInformation Element

The group of elements that is required for both classified and unclassified artifacts includes the classification code (U, C, or S), the export control determination (either Yes or No), and the distribution statement (with possible values of A, B, C, D, E, F, or X). These elements are shown for the *Unclassified* element in Figure 42.

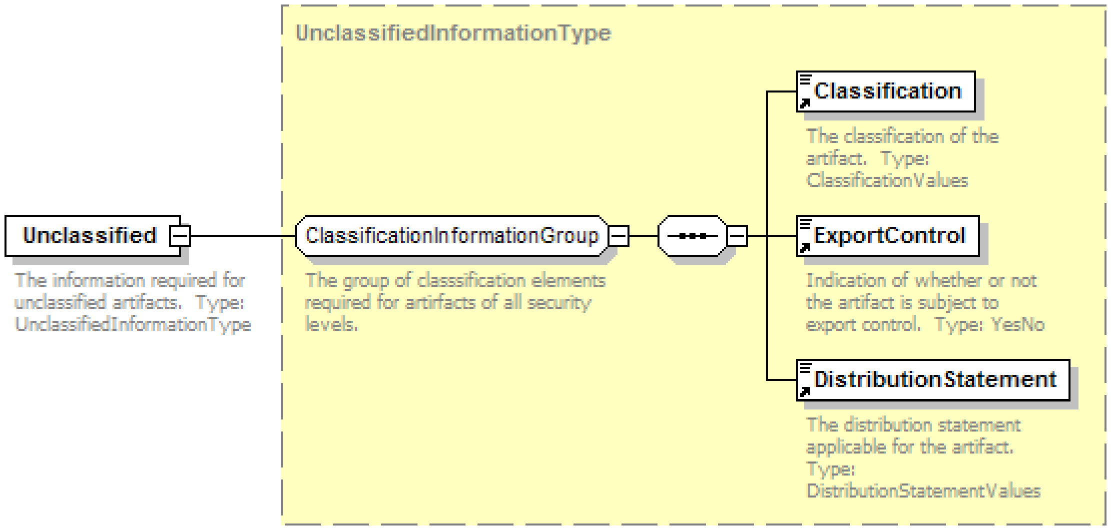

UnclassifiedInformationType
Classification
The classification of the artifact. Type: ClassificationValues
Unclassified
The information required for unclassified artifacts. Type: UnclassifiedInformationType
ClassificationInformationGroup
The group of classsification elements required for artifacts of all security levels.
ExportControl
Indication of whether or not the artifact is subject to export control. Type: YesNo
DistributionStatement
The distribution statement applicable for the artifact. Type: DistributionStatementValues

Figure 42. Unclassified Element

The *Classified* element includes each marking in addition to the classification guide ID, which cites the applicable DoD Classification Guide for the artifact. This is shown in Figure 43.

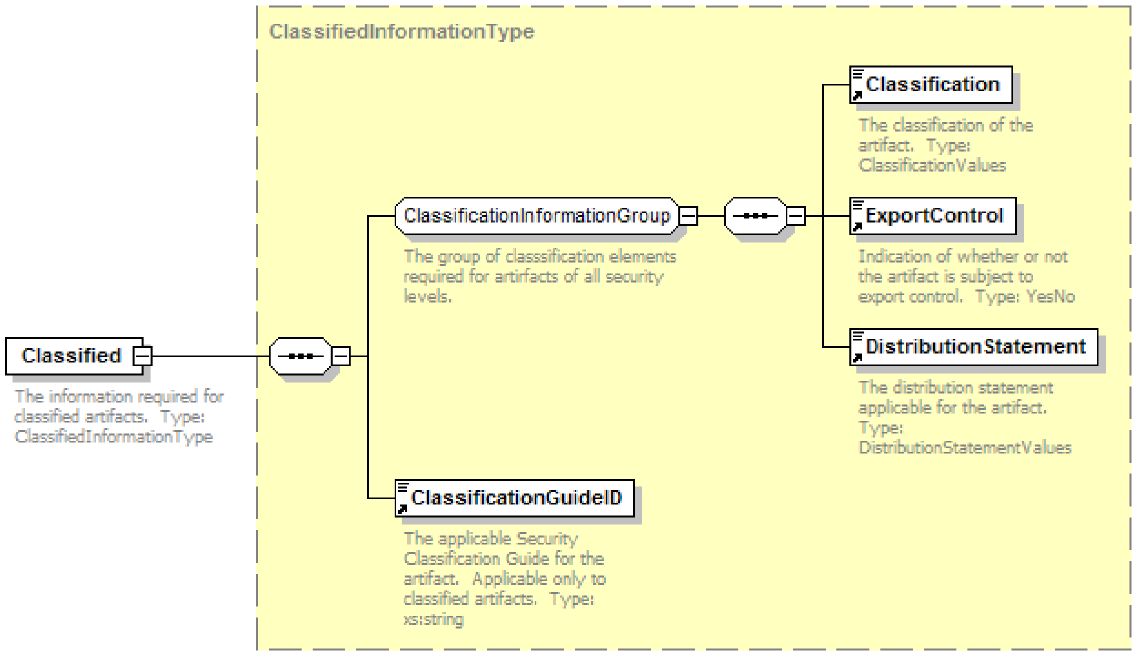

ClassifiedInformationType
Classified
The information required for classified artifacts. Type: ClassifiedInformationType
ClassificationInformationGroup
The group of classsification elements required for artirfacts of all security levels.
Classification
The classification of the artifact. Type: ClassificationValues
ExportControl
Indication of whether or not the artifact is subject to export control. Type: YesNo
DistributionStatement
The distribution statement applicable for the artifact. Type: DistributionStatementValues
ClassificationGuideID
The applicable Security Classification Guide for the artifact. Applicable only to classified artifacts. Type: xs:string

Figure 43. Classified Element

DataFormat

The *DataFormat* element refers to information about the format of the artifact and is depicted in Figure 44. Information is stored here about the type of physical media, which at this time may be CD or DVD. File names (one or many) for the artifact are provided here as well. This is the only mandatory entry in the *DataFormat* element. Archive formats and total data size are also included in the *DataFormat* element.

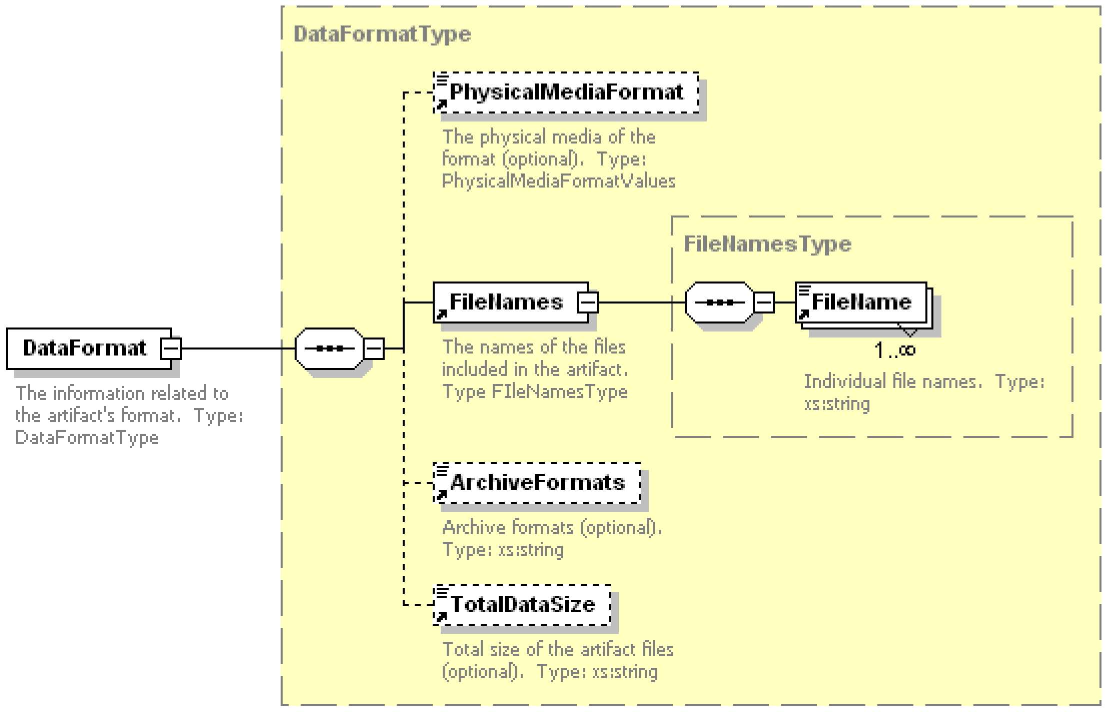

DataFormatType
PhysicalMediaFormat
The physical media of the format (optional). Type: PhysicalMediaFormatValues
FileNamesType
FileNames
The names of the files included in the artifact. Type FIleNamesType
FileName
1..∞
Individual file names. Type: xs:string
DataFormat
The information related to the artifact's format. Type: DataFormatType
ArchiveFormats
Archive formats (optional). Type: xs:string
TotalDataSize
Total size of the artifact files (optional). Type: xs:string

Figure 44. DataFormat Element

RightsRestrictions

The *RightsRestrictions* element contains information about possible limitations on the use of the artifact, as shown in Figure 45. For each subelement, the metadata contains a Yes or No indication of whether the restriction applies and then allows for a free text description of the restriction. The potential restrictions included are data rights markings, commercial software, special licenses, open source software licenses, and data rights assertions.

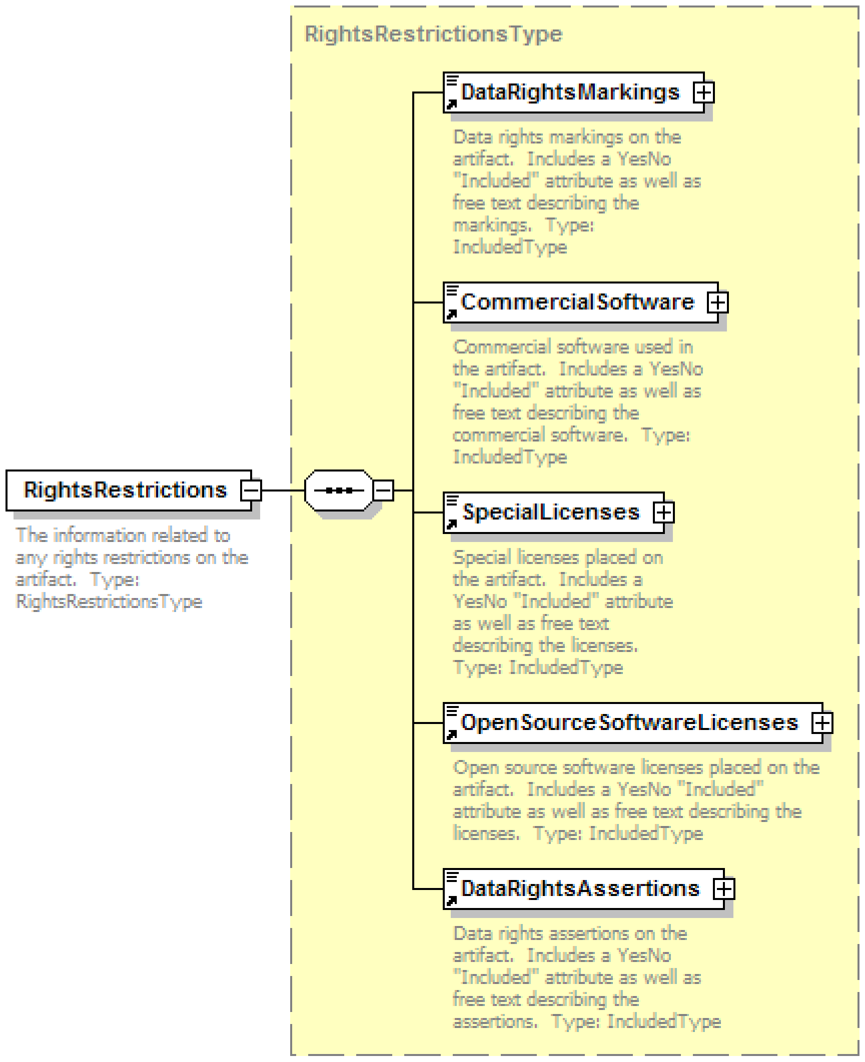

RightsRestrictionsType
DataRightsMarkings
Data rights markings on the artifact. Includes a YesNo "Included" attribute as well as free text describing the markings. Type: IncludedType
CommercialSoftware
Commercial software used in the artifact. Includes a YesNo "Included" attribute as well as free text describing the commercial software. Type: IncludedType
RightsRestrictions
The information related to any rights restrictions on the artifact. Type: RightsRestrictionsType
SpecialLicenses
Special licenses placed on the artifact. Includes a YesNo "Included" attribute as well as free text describing the licenses. Type: IncludedType
OpenSourceSoftwareLicenses
Open source software licenses placed on the artifact. Includes a YesNo "Included" attribute as well as free text describing the licenses. Type: IncludedType
DataRightsAssertions
Data rights assertions on the artifact. Includes a YesNo "Included" attribute as well as free text describing the assertions. Type: IncludedType

Figure 45. RightsRestrictions Element

AdditionalInformation

The final element in the full artifact description is an optional free text field, *AdditionalInformation*, in which an artifact submitter may enter any other relevant information about the artifact.

2. Assets Schema

In the preceding description of Artifacts, we see that much of the detail about a submission has been moved to the Artifact level. The information needed to describe an Asset is thus simplified to be primarily an identification of the Artifacts contained in the Asset. The root element of the Assets XML structure is a container for one or more Asset records, as shown in Figure 46. The proposed top-level XML structure for an Asset is shown in Figure 47.

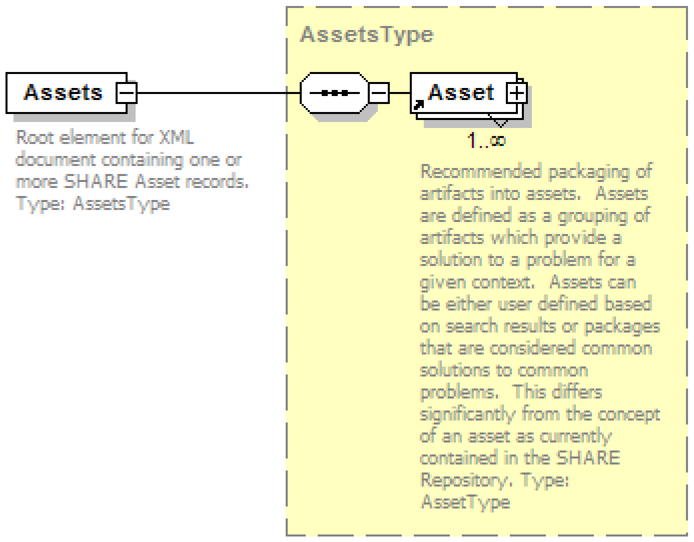

AssetsType
Assets
Root element for XML document containing one or more SHARE Asset records. Type: AssetsType
Asset
1..∞
Recommended packaging of artifacts into assets. Assets are defined as a grouping of artifacts which provide a solution to a problem for a given context. Assets can be either user defined based on search results or packages that are considered common solutions to common problems. This differs significantly from the concept of an asset as currently contained in the SHARE Repository. Type: AssetType

Figure 46. Assets Root Element

The Asset element provides the asset name, purpose, description of initial uses, history of uses (*PreviousUses*), scope (Tactical Application, Development Support, Other, or Unknown), category, list of contained artifacts (*ArtifactsIncluded*), and retrieval information. Of these, only *PreviousUses* and *ArtifactsIncluded* have subordinate structure (i.e., child elements). These structures are shown in Figure 38 and Figure 49, respectively. *PreviousUses* simply contains one or more *PreviousUse* elements, which contain a text description of the usage of the asset being submitted. *ArtifactsIncluded* is a type of *ArtifactsType*, a structure containing a list of one or more *Artifact* elements. This type, and the rest of the structure of an Artifact, was described previously in Section III.C.1.

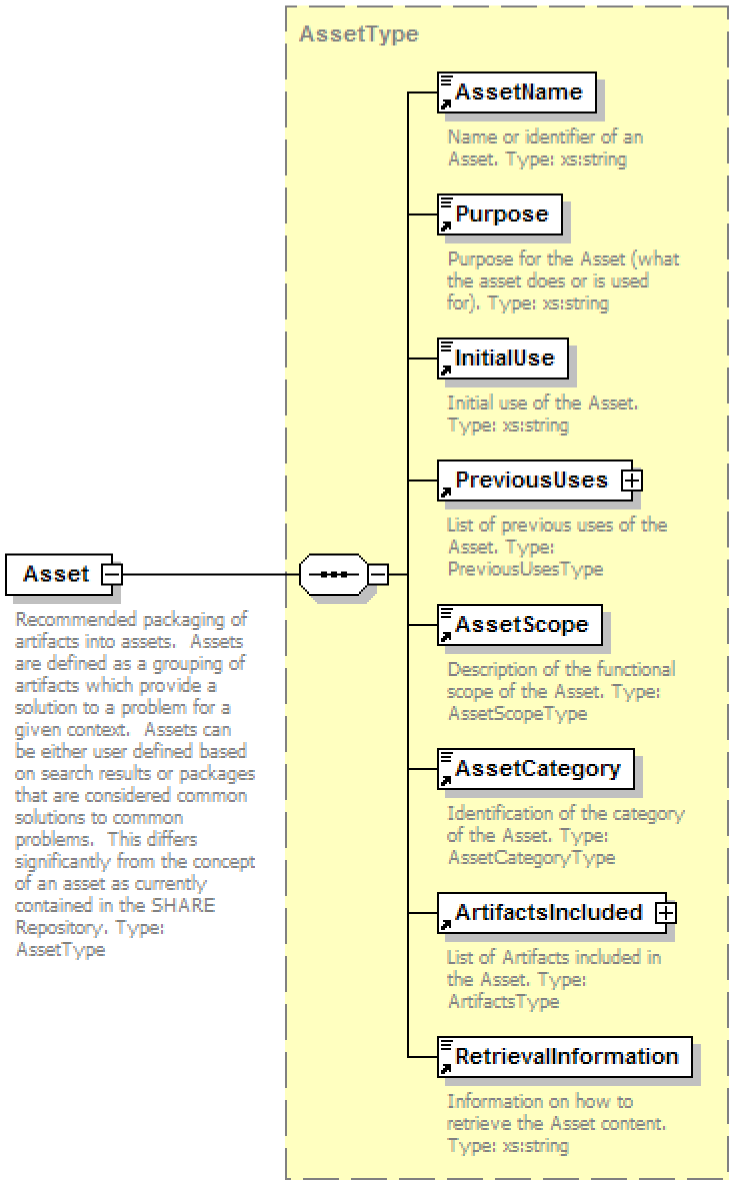

AssetType
AssetName
Name or identifier of an Asset. Type: xs:string
Purpose
Purpose for the Asset (what the asset does or is used for). Type: xs:string
InitialUse
Initial use of the Asset. Type: xs:string
PreviousUses
List of previous uses of the Asset. Type: PreviousUsesType
Asset
Recommended packaging of artifacts into assets. Assets are defined as a grouping of artifacts which provide a solution to a problem for a given context. Assets can be either user defined based on search results or packages that are considered common solutions to common problems. This differs significantly from the concept of an asset as currently contained in the SHARE Repository. Type: AssetType
AssetScope
Description of the functional scope of the Asset. Type: AssetScopeType
AssetCategory
Identification of the category of the Asset. Type: AssetCategoryType
ArtifactsIncluded
List of Artifacts included in the Asset. Type: ArtifactsType
RetrievalInformation
Information on how to retrieve the Asset content. Type: xs:string

Figure 47. Asset Element

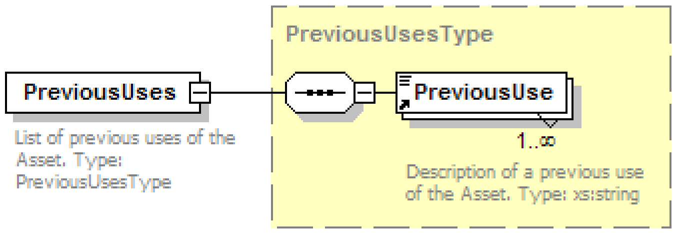

PreviousUsesType
PreviousUses
List of previous uses of the Asset. Type: PreviousUsesType
PreviousUse
1..∞
Description of a previous use of the Asset. Type: xs:string

Figure 48. PreviousUses Element

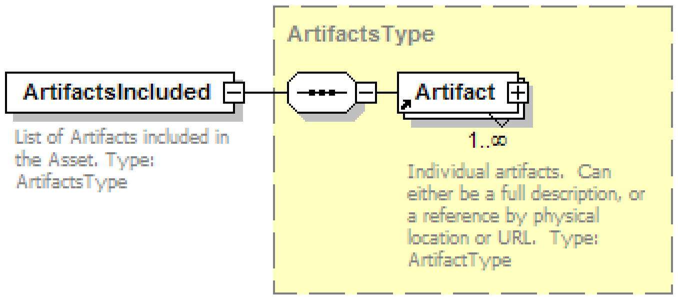

ArtifactsType
ArtifactsIncluded
List of Artifacts included in the Asset. Type: ArtifactsType
Artifact
1..∞
Individual artifacts. Can either be a full description, or a reference by physical location or URL. Type: ArtifactType

Figure 49. ArtifactsIncluded Element

D. Comparison of Current and Proposed Metadata

The proposed separation of Assets and Artifacts into their own XML schemas, including definition of global complex and simple types in the schemas, is intended to provide flexibility in using the schemas. Users can begin to describe individual artifacts prior to gathering collections of artifact descriptions to package into one or more assets that can be stored in the SHARE repository. Descriptive elements from the XML schemas (and therefore from the declared SHARE namespace) can be used in the construction of other XML schemas and XML documents, permitting common naming and data structures across applications.

Whether taking an asset-centric view or an artifact-centric view, greater automation is possible by having XML schemas available that show what information is needed to describe the resources. Submitter-side tools can extract needed information from project files to generate XML files containing resource descriptions and can submit those to the SHARE repository for processing and storage.

In accessing information from the SHARE repository, the artifact-centric perspective enables users to be more precise in their searches and to obtain more precise responses. Previously, access at the asset level meant a request could include unneeded or unwanted portions of an asset, where only certain selected artifacts within the asset were of interest. At the artifact level, responses to user requests can be tailored more specifically to users' needs. On the one hand, if releasability authorization is at the artifact level, there is stronger control over the release of a component, but multiple release authorities may create more difficulty in obtaining the components needed. We think erring on the side of stronger control and precision is worth the trade-off with the potential of a more difficult access to the actual product. Much of this difficulty may be overcome, in time, through various access control methods that permit greater automation in the release authorization process.

In addition to this fundamental difference between the existing and proposed metadata sets, there were several changes to the structure and details included in the metadata. Additional changes could be desirable based on an evaluation of the usefulness of some of the existing metadata fields.

E. Metadata Comparison with Existing Repositories

To verify completeness of the metadata, we conducted a brief analysis in which we compared the proposed metadata to the information captured in two known software repositories, SourceForge (2007) and CPAN (2007). Although these repositories have key differences from SHARE in content, purpose, and structure, they serve as examples of working software repositories and are used here as a “sanity check” for the completion of the metadata.

SourceForge provides both an open source software repository and a project management tool. The repository contains downloadable software from the projects stored at the site. SourceForge enables essentially two different ways to search. First, users can browse the repository by clicking through categorizations of different types of software and then they can refine the search by filtering for different program aspects, such as specific program language or operating system. Second, a keyword search over the metadata within a particular category is possible. Compared to other online repositories, the metadata in SourceForge is quite exhaustive. This is due, in part, to the convenience of drawing the metadata from the project information at the same location.

In general, the proposed SHARE metadata is similar to the information posted about SourceForge artifacts. The most significant differences reflect the different uses of the repositories. Since SourceForge is also intended to be an interactive developer environment rather than just a repository of complete artifacts, there are different fields introduced to capture some of the information to help the developers. For example, SourceForge allows the developer to establish forums for users to submit feature requests, support requests, and to ask for help when using the software. In SHARE, it may be desirable for users to blog or to have other capabilities attached to the artifacts or assets, but it is unlikely that the same level of attention will be paid to user requests since the requirements for the software and any upgrades are generated through the DoD acquisition process.

CPAN also contains similar information about its artifacts to that found in the SHARE repository. CPAN seems to be focused at a much lower level of granularity, however. For example, when describing software behavior, CPAN lists the methods included in the software package and describes each of them. This can be thought of as a more detailed version of the proposed CSFL descriptions of the software behavior or similar to the WSDL descriptions discussed in the next section. Both are capturing software functionality, but the CSFL descriptions are held at a higher level of abstraction than the individual methods described in CPAN.

CPAN and SourceForge both contain a metadata structure for known errors of a system. We considered adding this field to the SHARE metadata, but it is likely that a collection of known faults of secret systems is sensitive information. This could easily be added at a later time if deemed desirable; however, change management is probably best handled by the individual programs following their established processes and procedures.

F. DDMS Element Mapping

In the US Department of Defense, the guiding document for information sharing is the Net-Centric Data Sharing Strategy (DoD Chief Information Officer, 2003). Under this strategy, the DoD Discovery Metadata Specification (DDMS) (Deputy Assistant Secretary of Defense, 2007) provides a standard set of metadata for discovering distributed resources across the DoD Enterprise.

To address this directive, we performed an analysis to ensure that sufficient metadata are provided in the descriptions of assets and artifacts to allow generation of at least the minimum required set of metadata specified in the DDMS. The mapping of DDMS elements to SHARE metadata elements is presented in Table 3. The DDMS elements are listed on the left and the corresponding SHARE elements on the right.

Table 3. DDMS to SHARE Element Mapping

DDMS Primary Category Sets			SHARE Schema Mapping (1)	
Core Layer Category Set	**Primary Category**	**Obligation**	**Current SHARE Metadata Elements (“as-is” schema)**	**Proposed SHARE Metadata Elements (Artifact level)**
The **Security** elements enable the description of security classification and related fields	Security	Mandatory	AssetDescription/ClassificationInforrmation	ArtifactFullDescription / SecurityInformation
Resource elements enable the description of maintenance and administrative information	Title	Mandatory	AssetDescription/AssetName AssetDescription/Version	ArtifactFullDescription/ ArtifactDescription/ (NonCodeDescription or CodeDescription)/ ArtifactName ArtifactFullDescription/ ArtifactDescription/ (NonCodeDescription or CodeDescription)/Version
	Identifier	Mandatory	ProgramInformation/Program	ArtifactFullDescription/ ControlNumber
	Creator	Mandatory	SourceIdentification/Contributor	ArtifactFullDescription/ SubmissionInformation/ Source/ArtifactSource
	Publisher	Optional	N/A (Publisher is SHARE Program Office)	N/A (Publisher is SHARE Program Office)
	Contributor	Optional	SourceIdentification/Organization	ArtifactFullDescription/ SubmissionInformation/ Source/ SourceOrganization
	Date	Optional	AssetDescription/Date	ArtifactFullDescription/ SubmissionInformation/ SubmitDate ArtifactFullDescription/ SubmissionInformation/ SubmitTime ArtifactFullDescription/ ArtifactDescription/ (NonCodeDescription or CodeDescription)/ DateOfCreation
	Rights	Optional	RightsAndRestrictions	ArtifactFullDescription/ RightsRestrictions
	Language	Optional	N/A (English Assumed)	N/A (English Assumed)
	Type	Optional	AssetDescription/TypeAsset	ArtifactFullDescription/ ArtifactDescription/ (NonCodeDescription or CodeDescription)/ ArtifactType

DDMS Primary Category Sets			SHARE Schema Mapping (1)	
Core Layer Category Set	**Primary Category**	**Obligation**	**Current SHARE Metadata Elements ("as-is" schema)**	**Proposed SHARE Metadata Elements (Artifact level)**
	Source	Optional	N/A (Covered by Creator and Contributor)	N/A (Covered by Creator and Contributor)
The **Summary Content** elements enable the description of concepts and topics	Subject	Mandatory	AssetScope/ (TacticalApplication or DevelopmentSupport or OtherScope or UnknownScope)	ArtifactFullDescription/ ArtifactDescription / (NonCodeDescription or CodeDescription)/ ObjectiveArchitectureTags
	Geospatial Coverage	Mandatory unless not Applicable	N/A	N/A
	Temporal Coverage	Mandatory unless not Applicable	N/A	N/A
	Virtual Coverage	Optional	N/A	N/A
	Description	Optional	AssetDescription/Description	ArtifactFullDescription/ ArtifactDescription/ (NonCodeDescription or CodeDescription)/ Description
The **Format** elements enable the description of physical attributes of the asset	Format	Optional	DataFormat	ArtifactFullDescription/ DataFormat

NOTES:
(1) Mapping to the schema structure shows the element hierarchy down to the element of interest. For brevity, the "Current SHARE Metadata Elements" column leaves out the root *SHAREAssetInformation* element in the hierarchy and the "Proposed SHARE Metadata Elements" column leaves out the upper two elements *Artifacts*/*Artifact* of the hierarchy.

G. Summary

The ability to provide metadata describing SHARE resources (assets and artifacts) in XML documents conforming to XML schemas is the first step toward achieving the vision of the SHARE repository framework. The XML structures create new opportunities for advanced automation in the specification and discovery of resources to meet users' needs. The structured metadata facilitates data extraction to support discovery of SHARE resources in the broader DoD context through emerging net-centric data sharing technologies.

IV. SHARE Software Behavior Representation

A. Introduction

One of the loftier goals of a software repository is to support automatic composition of systems from reusable components. This is a difficult problem, which many have tried to solve.[10] It is especially difficult if the components were not originally designed for reuse. As a necessary first step towards more sophisticated uses of a repository, behavioral descriptions must be machine-readable in order to support automated search and discovery. Furthermore, the behavior descriptions must be formalized and consistently applied to each item in the repository if the intent is to automatically compose them into a larger functioning system.

In this section, we describe initial work toward standardized specification of software behavior for the SHARE repository. Each type of presented representation offers advantages for certain purposes. However, it is recognized that the array of contributors to SHARE requires caution in dictating standards that will impact the development processes of the asset developers. We have sought a balance between method robustness and ease of implementation for this phase of the effort.

The formalized description of software behavior provides two vital pieces of information: (1) inputs and outputs (interfaces) of the components; and (2) functionality provided by the component.

B. Interface Descriptions

Interface descriptions focus on the inputs and outputs of a component and not the inner workings of that component. Interfaces are represented using various

[10] The proceedings from the International Symposium on Software Composition, an annual event, provide examples of research into the breadth of research topics currently being pursued in the area of software composition. The web site for the 2008 conference is located at http://www.2008.software-composition.org/

methods, which vary from concentration on the connect points between two pieces of software and the types of information passed between them to representations of the services that a component provides.

Navy systems are evolving toward the Service-Oriented Architecture (SOA) of the GIG. SOA has been described as “an ideal vision of a world in which resources are cleanly partitioned and consistently represented” (Erl, 2005, p. 3) and “automation logic is decomposed into smaller distinct units of logic … known as *services*” (pp. 23-33). Elements of a service architecture are similar to SHARE concerns—the architecture typically includes a registry of services containing descriptions of those services and information on how to access them. Mechanisms are provided for service discovery and for passing sufficient information about the service back to the caller so that the service can be invoked. Advanced concepts include service orchestration for composing higher order services from component services. The focus, of course, is service reuse, which will potentially reduce development and maintenance while improving software reliability and evolution agility. These concepts clearly align with objectives of the SHARE repository in which components (although not limited to software) are also made available for reuse.

SOA realization employs a variety of standards, including Web Services standards such as: Universal Description, Discovery, and Integration (UDDI)[11] for creating service registries; Web Services Description Language (WSDL)[12] for identifying operations offered by services and describing input/output interfaces for those operations; the Simple Object Access Protocol (SOAP)[13] for accessing services and passing data to/from the services; Web Services Business Process

[11] For UDDI information, see: http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm
[12] For WSDL information, see: http://www.w3.org/2002/ws/desc/
[13] For SOAP information, see: http://www.w3.org/TR/soap/

Execution Language (WS-BPEL)[14] for describing workflow logic for orchestration of services; OWL for Services (OWL-S),[15] an ontology of services supporting service advertisement and discovery, description of service operation, and service interoperation; Web Services Interoperability (WS-I) profiles[16] describing collections of Web services specifications at specific version levels; and others. Again, it is interesting to note that the problem of describing Web services in sufficient semantic detail to enable automatic composition of services is similar to the problem of describing software components in SHARE for reuse.

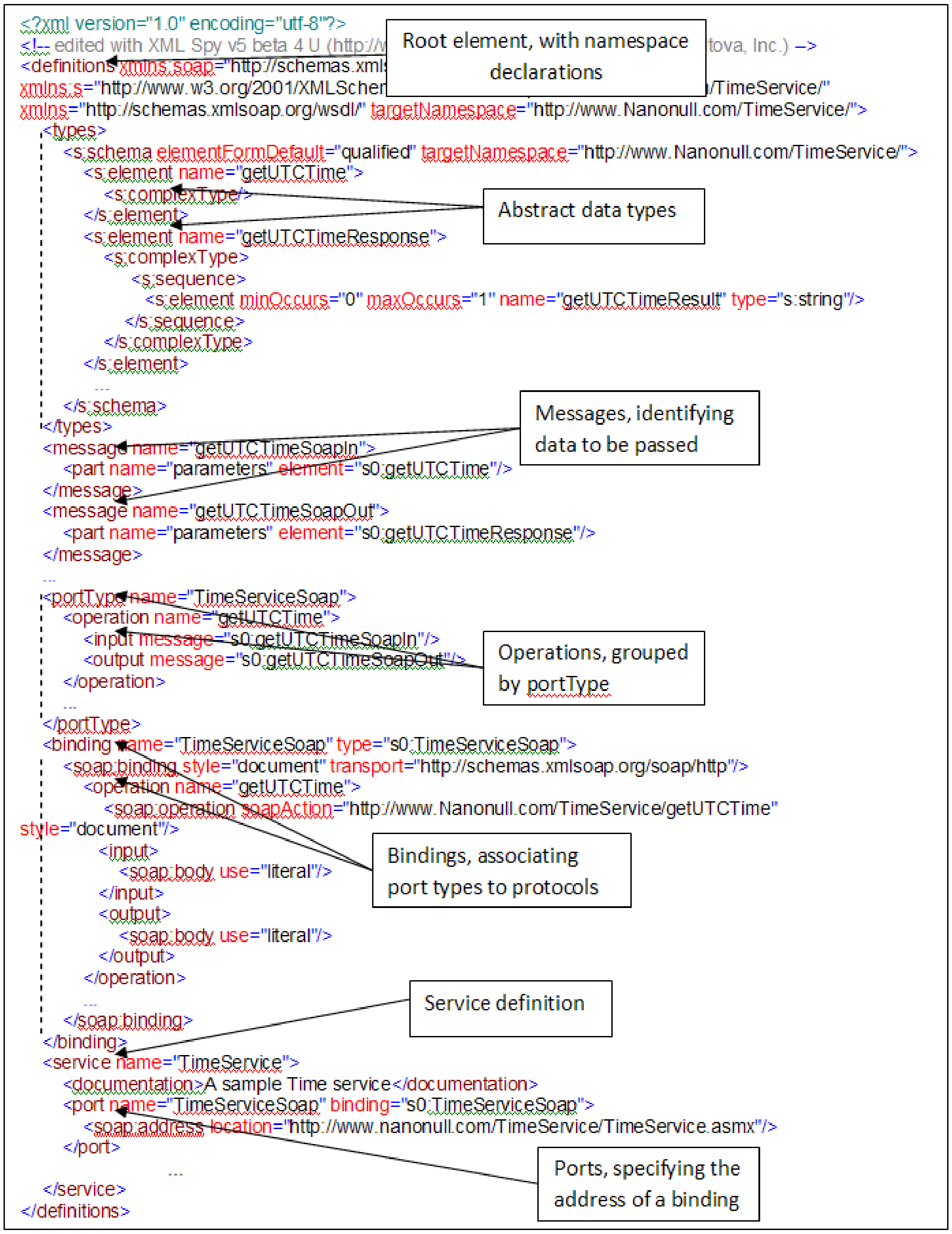
In this research, we explored characterization of software interfaces based on current and emerging Web Services (e.g., WSDL) and Semantic Web Services (e.g., WS-BPEL, OWL-S) approaches. However, the work is preliminary, since the current approach to describing code artifacts making up an asset is extremely limited, as we saw in the previous chapter. It will be necessary to adopt a more precise description of code artifacts to introduce these techniques. As a start, we included the option of inserting a WSDL description of software services in the *SoftwareBehaviorDescription* element described in section III.C.1 (see also Figure 36).

A WSDL document is an XML file that describes information needed to invoke a Web service. This information includes: (1) abstract data *types* (specified by XML Schema language or references to XML schemas) used in interactions with operations offered by the service; (2) *messages* that define data structures (using the abstract data types above) for interactions with operations offered by the service; (3) *operations* identifying the specific actions the service will perform and the input and output messages associated with the operations; (4) *port types* that group a number of operations for binding to specific protocols (e.g., SOAP, HTTP); (5)

---

[14] For WS-BPEL information see: http://www.ibm.com/developerworks/library/specification/ws-bpel/
[15] For OWL-S information, see: http://www.w3.org/Submission/OWL-S/
[16] For WS-I information, see: http://www.ws-i.org/

bindings associating port types to protocols; (6) *ports* specifying the address of a binding; and (7) the service definition combining the ports defined above (Arora & Kishore, 2002). Excerpts of a WSDL document illustrating these parts of the service specification are shown in Figure 50.


Root element, with namespace declarations
Abstract data types
Messages, identifying data to be passed
Operations, grouped by portType
Bindings, associating port types to protocols
Service definition
Ports, specifying the address of a binding

```
<?xml version="1.0" encoding="utf-8"?>
<!-- edited with XML Spy v5 beta 4 U (http://w                                tova, Inc.) -->
<definitions xmlns:soap="http://schemas.xml
xmlns:s="http://www.w3.org/2001/XMLScher                                      /TimeService/"
xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://www.Nanonull.com/TimeService/">
  <types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://www.Nanonull.com/TimeService/">
      <s:element name="getUTCTime">
        <s:complexType/>
      </s:element>
      <s:element name="getUTCTimeResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="getUTCTimeResult" type="s:string"/>
          </s:sequence>
        </s:complexType>
      </s:element>
      ...
    </s:schema>
  </types>
  <message name="getUTCTimeSoapIn">
    <part name="parameters" element="s0:getUTCTime"/>
  </message>
  <message name="getUTCTimeSoapOut">
    <part name="parameters" element="s0:getUTCTimeResponse"/>
  </message>
  ...
  <portType name="TimeServiceSoap">
    <operation name="getUTCTime">
      <input message="s0:getUTCTimeSoapIn"/>
      <output message="s0:getUTCTimeSoapOut"/>
    </operation>
    ...
  </portType>
  <binding name="TimeServiceSoap" type="s0:TimeServiceSoap">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="getUTCTime">
        <soap:operation soapAction="http://www.Nanonull.com/TimeService/getUTCTime"
style="document"/>
          <input>
            <soap:body use="literal"/>
          </input>
          <output>
            <soap:body use="literal"/>
          </output>
      </operation>
      ...
    </soap:binding>
  </binding>
  <service name="TimeService">
    <documentation>A sample Time service</documentation>
    <port name="TimeServiceSoap" binding="s0:TimeServiceSoap">
      <soap:address location="http://www.nanonull.com/TimeService/TimeService.asmx"/>
    </port>
              ...
  </service>
</definitions>
```

Figure 50. Excerpts from a Sample WSDL Document Showing Principal Components of the Service Specification (example from Altova XML-Spy)

As the DoD moves toward SOA, services may become a more frequent part of the SHARE repository. In that case, the WSDL describing those services (often automatically generated by the software development or execution environment of modern software systems) can be directly utilized in the repository to provide a detailed view of the service interfaces and operations. For software that is not developed and deployed as services, it is still feasible for public methods within the software to be parsed automatically to create WSDL-like descriptions. These would likely be incomplete descriptions with respect to full compliance to WSDL structures, but could still provide a well-defined way to describe the software for search and discovery. However, the question remains, "What level of decomposition of software components, with associated description, is appropriate for the SHARE repository?" More research is needed in this area to address this and other questions for practical application in SHARE.

Moreover, WSDL alone is not sufficient to enable automated discovery and composition of services, let alone more general software components. Semantic Web Services is a research area attempting to address this issue, but much remains to be accomplished. While WSDL identifies operations within a service and provides sufficient information for invoking the operations, it does not describe the functionality offered in and performed by those operations. This, too, is a major area of research. In the next section, we describe an initial approach using a well-established taxonomy of functionality relevant to the Navy combat system architecture.

C. Describing Functionality

In addition to understanding the interfaces for a component, a repository user is interested in the functionality of the software components. We propose a near-term solution that uses domain information to standardize descriptions of software functionality; namely, the well-established CSFL.[17] We developed a taxonomy based

[17] DoD Warfighter Service Components in the DoD Enterprise Architecture Service Component Reference Model are derived from the DoN CSFL.

on the CSFL and incorporated fields into the metadata (XML schema) that will assign functions to repository items. If we require asset submitters to state the functionality of the components in these terms, we can then build the tools to guide users in selecting desired behavior in the same terms.

The CSFL was captured in an OWL structure to use as an initial characterization of software behavior. The process by which the taxonomy was generated is a good example of methods for creating a practical set of structured data from initial raw formats. The taxonomy was constructed from a Microsoft Excel spreadsheet listing the domains and functions within each domain (CSFL version 3.0). The spreadsheet provided definitions of the domains and functions, identified what the domain or function is derived from and identified sources of the definitions. Microsoft Excel provides the capability to export the content of the spreadsheet to XML format. A simple Extensible Stylesheet Language for Transformations (XSLT) was written to transform the source XML format (spreadsheet data) to a target XML format (OWL). The transformation created a simple class/subclass hierarchy expressed in OWL. A portion of the resulting OWL structure is shown in the Protégé ontology editing tool in Figure 51.

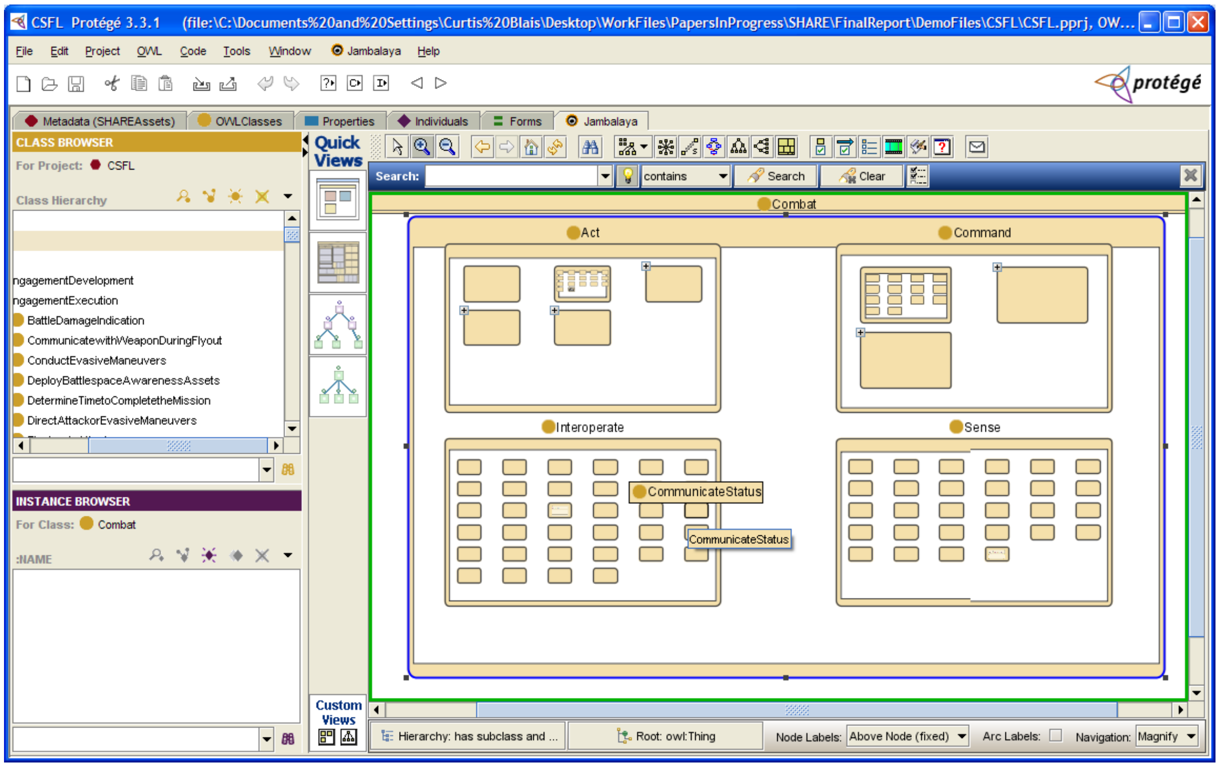

CSFL Protégé 3.3.1 (file:\C:\Documents%20and%20Settings\Curtis%20Blais\Desktop\WorkFiles\PapersInProgress\SHARE\FinalReport\DemoFiles\CSFL\CSFL.pprj, OW...
File Edit Project OWL Code Tools Window Jambalaya Help
Metadata (SHAREAssets) OWLClasses Properties Individuals Forms Jambalaya
CLASS BROWSER
For Project: CSFL
Class Hierarchy
ngagementDevelopment
ngagementExecution
BattleDamageIndication
CommunicatewithWeaponDuringFlyout
ConductEvasiveManeuvers
DeployBattlespaceAwarenessAssets
DetermineTimetoCompletetheMission
DirectAttackorEvasiveManeuvers
INSTANCE BROWSER
For Class: Combat
:NAME
Quick Views
Custom Views
Search: contains Search Clear
Combat
Act
Command
Interoperate
Sense
CommunicateStatus
CommunicateStatus
Hierarchy: has subclass and ...
Root: owl:Thing
Node Labels: Above Node (fixed)
Arc Labels:
Navigation: Magnify

Figure 51. Portion of the CSFL Taxonomy Displayed in Protégé under the Jambalaya Graphics Tab

Other similar lists have been developed for operational activities (i.e., the Common Operational Activities List [COAL]) and for information elements (Common Information Element List [CIEL]). It may be valuable to also capture these in OWL classes and then to create interrelationships across the classes (e.g., what information elements are generally employed in performing certain system functions and what information elements are generally produced by performing certain system functions, etc.). Further exploration with subject matter experts is needed to determine potential benefit from such approaches.

D. Summary

Although we cannot solve the software composition problem in the near term, initial descriptions of software behavior through identification of functionality and specification of interfaces are necessary steps toward that capability. These intermediate steps towards formalized behavior descriptions will prove useful in the near term and helpful in advancing towards far-term goals. As shown in section III, the WSDL descriptions and functionality identification can be integrated into XML descriptions of the artifacts as standardized behavioral descriptions for each artifact entered into the repository. Future work should address ongoing advances in service composition in SOAs for application to the framework and automated generation of such information from the artifact.

THIS PAGE INTENTIONALLY LEFT BLANK

V. SHARE Relationships Framework (Ontology)

A. Introduction

Assets and artifacts in the SHARE repository can be examined from a number of different perspectives, reflecting a variety of associations. We chose to create initial classification schemes that can provide benefit in the near term. The resulting taxonomies and ontologies are meant to be illustrative not exhaustive. The taxonomies/ontologies we developed for SHARE are based on several types of relationships between the items in the repository, as well as with relevant domain architectural descriptions and other information. They capture an artifact's place in the software engineering lifecycle, its architectural fit in its original system, its architectural fit in any system in which it was subsequently used, identification of the component's fit in the Surface Navy Objective Architecture, and the semantic relationships of various documents in the repository (based on the ReSEARCH work). Each of these ontologies is discussed in further detail in the following sections.

We used Stanford's Protégé-OWL (Stanford, 2008) tool to develop the taxonomies/ontologies. This is an open source, free ontology editor, available online[18]. Users will be the repository software designers and the software agents built to query the repository for information. Maintenance of the ontologies will likely be the responsibility of the SHARE PM and staff.

B. Relating Artifacts to Lifecycle Phases

This ontology covers the domain of software artifacts and their relations to the software engineering lifecycle. The use of the ontology will enable the software repository to correlate artifacts that have similar relationships to suggest them as

[18] http://protege.stanford.edu/

possible items for retrieval. The types of questions for which the information in the ontology should provide answers include:

- If I am interested in a certain set of artifacts, I may also be interested in artifacts that are similar. So, if I am looking at an artifact of a certain type, what other artifacts are documents of the same type?
- I may also be interested in a particular set of artifacts depending on the lifecycle phase I am in when I come to the repository. What types of artifacts are commonly needed or useful when I am in a certain phase of the lifecycle?

A few notes about the conventions we used in the creation of the ontology follow:

- Class names at the node level are singular. This is somewhat arbitrary and only stipulated for consistency, but it enables the decision-maker to easily ask the question, “Is this artifact a(n) class name?” to help determine which class an item should belong to.
- To avoid confusion, items in the first layer of subclasses under the software artifacts class are assigned the suffix “Artifacts,” and those in the first layer of subclasses under the lifecycle phases are assigned the suffix “Activity.”
- Each word of a class name is capitalized with no delimiter or space between words (UpperCamelCase). The first word only of a property is lower case with no delimiter or space between words (lowerCamelCase).

1. Classes and Class Hierarchy

There are two main classes in the ontology—software artifacts and lifecycle activities. Each class is defined by a taxonomy of subclasses. These classes and the correlations of various relationships between the taxonomies form the ontology.

The software artifact taxonomy is shown in Figure 52. While depicted as a diagram here, it is also represented in table form in Table 1.

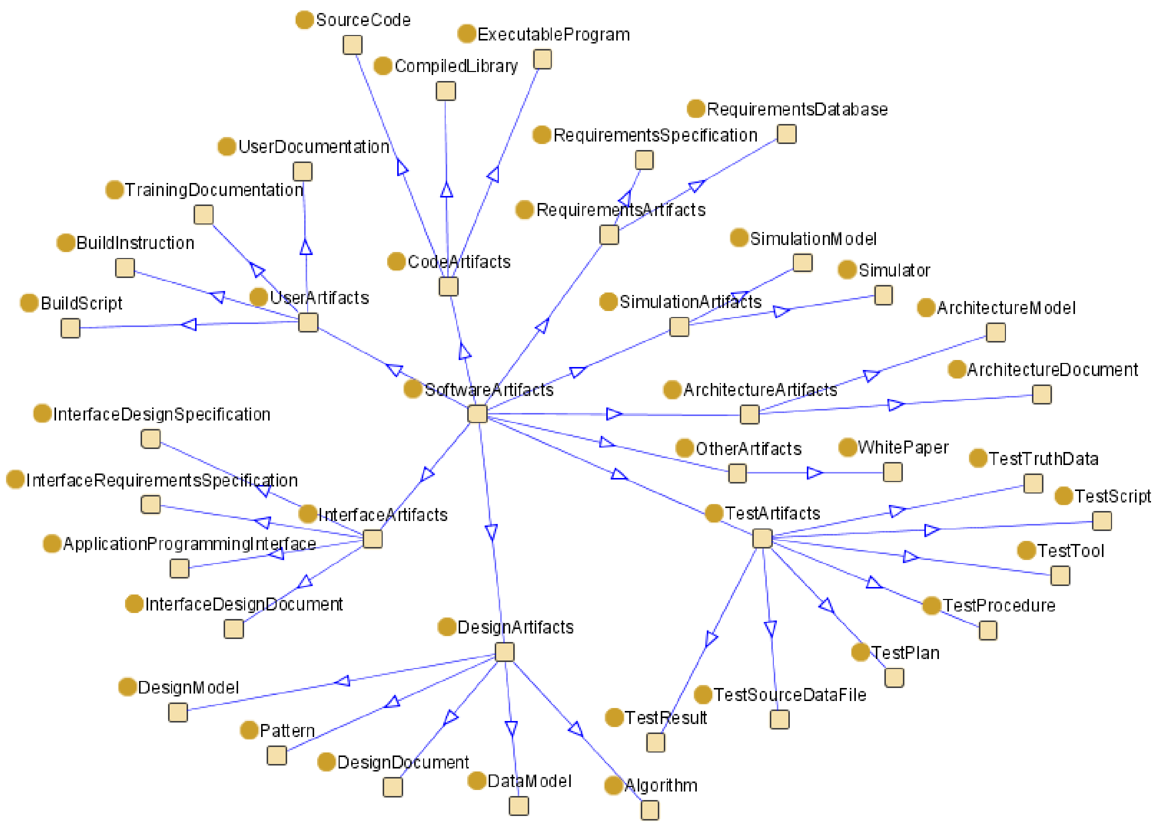


Figure 52. Software Artifact Taxonomy

The software artifact taxonomy is based on the artifact types provided in SHARE Asset Contribution spreadsheet (v6) developed by the SHARE program office at Dahlgren. However, changes to the original taxonomy include:

1. Distinguished between “grouped” items—for example, the original list had “Test Tools/Scripts” as one item, and we broke them out into two. We did the same for “IRSs/IDDs” and “Build Scripts/Instructions.”

2. Renamed “Supporting Artifacts” to “User Artifacts” since the former name seemed vague and each of the subclasses in the class are really user aids.

3. Made “Other” a separate subclass under Software Artifacts rather than a subclass of Supporting Artifacts. Either all subclasses should contain an Other category, or there should be one Other category for the entire class. We felt that there should be one location for any situation where the existing subclasses do not apply. Over time, items

in the Other category can be grouped into subclasses and the ontology modified as appropriate.

4. Moved white papers into the Other class. White papers could really be documentation of any decisions made about the system at any point in the lifecycle.

5. Created a new artifact type called SimulationArtifact. Previously, a simulation model was classified as a design artifact and a simulator as a test artifact. Depending on the intended use of the model or simulation, these categorizations are probably too restrictive.

The lifecycle phases taxonomy is presented in Figure 53. Since lifecycle phases and their assumed order are somewhat dependent on the development model in use (i.e., waterfall, spiral, incremental, etc.), we use “activities” to reflect the most generic possible classifications of the phases.

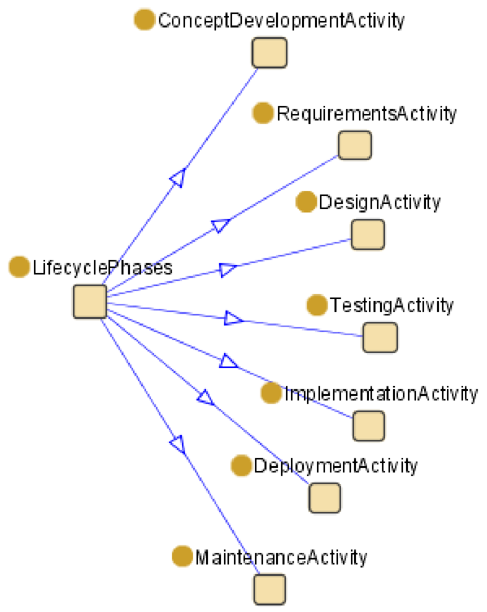


Figure 53. Lifecycle Phases Taxonomy

It is worth noting that the two top-level classes (Software Artifacts and Lifecycle Phases) contain similarly named subclasses at the first level. This is evidence that the artifacts produced during software development are often associated with the phase of their development—a relationship that this ontology aims to capture.

2. Class Properties

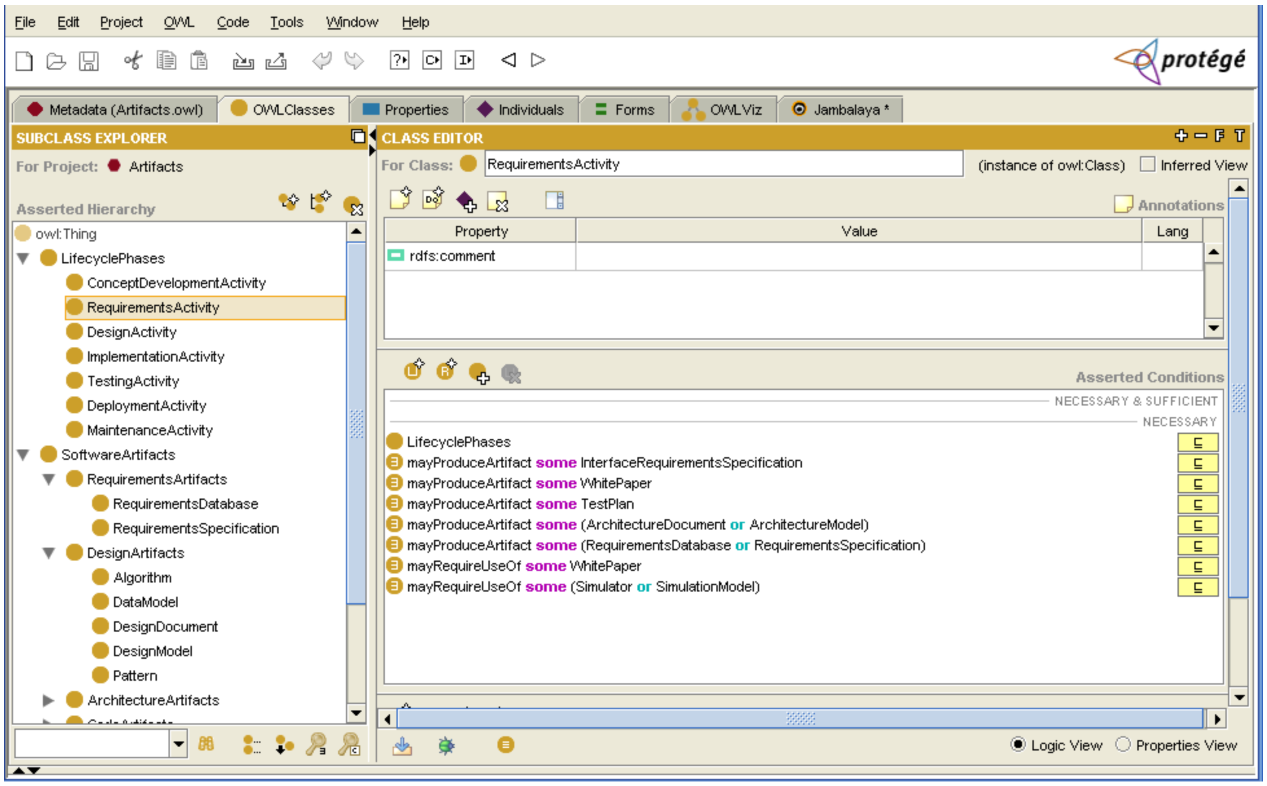
Properties are relationships that link items in and across classes. The properties in the artifact-lifecycle ontology connect artifact types to lifecycle activities through various relationships. An example of a property is the “has subclass” relationship between members of a taxonomy. This property captures the hierarchical relationships between classes and subclasses. Aside from the “has subclass” relationship that exists in the software artifacts and lifecycle activities taxonomies, there are four additional properties that link these class structures, including:

- mayProduceArtifact—For each lifecycle activity, identifies which artifacts are most commonly produced as a result of that activity. The inverse property is oftenDevelopedDuring. The property maps items in the LifecyclePhases class (domain of the property) to the SoftwareArtifact class (range of the property).
- oftenDevelopedDuring—For each artifact, identifies the activity or activities that most commonly produce it. The inverse property is mayProduceArtifact. The property maps items in the SoftwareArtifact class (domain) to the LifecyclePhases class (range).
- mayRequireUseOf—For each lifecycle activity, identifies the most commonly needed artifacts. The inverse property is oftenUsedDuring. The property maps items in the LifecyclePhases class (domain) to the SoftwareArtifact class (range).
- oftenUsedDuring—For each artifact, identifies the activity or activities in which it is most commonly needed. The inverse property is mayRequireUseOf. The property maps items in the SoftwareArtifact class (domain) to the LifecyclePhases class (range).

The assignment of the properties to the classes is somewhat subjective. Here we summarize the approach for assignment of each property type:

- mayProduceArtifact—For each lifecycle activity, the typical resulting artifacts are identified by this relationship.
- oftenDevelopedDuring—For each type of artifact, this relation assigns the lifecycle activities that are most likely to result in the artifact type. In some cases, it is straightforward, such as requirements specifications most likely being developed during the requirements activities. In other cases, there are multiple possibilities. For example, test plans could be developed at any point once the requirements are developed. In fact, many approaches to software engineering encourage the early development of test cases. Note that these first two relationships are most useful to find similar artifacts (of like or unlike systems) to those that you would need to develop during each phase of development.
- mayRequireUseOf—For each lifecycle activity that a developer might be executing, this property identifies the artifacts that may be useful to him/her.
- oftenUsedDuring—For each artifact type, this relation is assigned to relate the artifact to the lifecycle phases is it most commonly used in. Note that these last two relationships are useful if you are looking for particular items (of the same or a similar system) to those you are currently working on that would be useful during a development activity.

The complete ontology is provided in project deliverable materials. A few examples are provided here to give a sense for structure of the ontology. For example, the assignment of properties (relationships) for the RequirementsActivity class in the lifecycle phases taxonomy is presented in Figure 54.



For Project: Artifacts

owl:Thing
- LifecyclePhases
  - ConceptDevelopmentActivity
  - RequirementsActivity
  - DesignActivity
  - ImplementationActivity
  - TestingActivity
  - DeploymentActivity
  - MaintenanceActivity
- SoftwareArtifacts
  - RequirementsArtifacts
    - RequirementsDatabase
    - RequirementsSpecification
  - DesignArtifacts
    - Algorithm
    - DataModel
    - DesignDocument
    - DesignModel
    - Pattern
  - ArchitectureArtifacts

For Class: RequirementsActivity (instance of owl:Class) Inferred View

Property	Value	Lang
rdfs:comment		

LifecyclePhases
mayProduceArtifact **some** InterfaceRequirementsSpecification
mayProduceArtifact **some** WhitePaper
mayProduceArtifact **some** TestPlan
mayProduceArtifact **some** (ArchitectureDocument **or** ArchitectureModel)
mayProduceArtifact **some** (RequirementsDatabase **or** RequirementsSpecification)
mayRequireUseOf **some** WhitePaper
mayRequireUseOf **some** (Simulator **or** SimulationModel)

Figure 54. Properties Assigned to RequirementsActivity Class

Classes and their relationships can also be presented as a diagram, as shown for the RequirementsSpecification and RequirementsDatabase classes of the software artifact taxonomy in Figure 55. This diagram was generated by Jambalaya (CHISEL, 2008), an open source plug-in for Protégé.

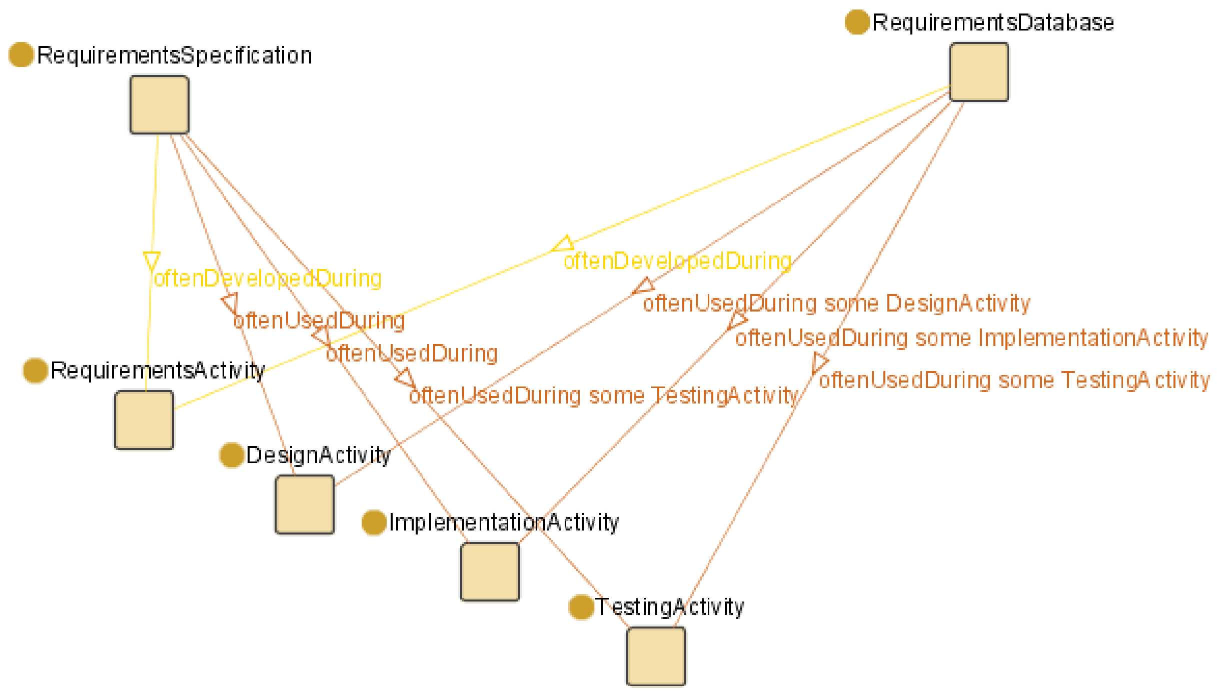

RequirementsSpecification
RequirementsDatabase
oftenDevelopedDuring
oftenUsedDuring
oftenUsedDuring
oftenUsedDuring some TestingActivity
oftenDevelopedDuring
oftenUsedDuring some DesignActivity
oftenUsedDuring some ImplementationActivity
oftenUsedDuring some TestingActivity
RequirementsActivity
DesignActivity
ImplementationActivity
TestingActivity

Figure 55. Properties Assigned to RequirementsSpecification and RequirementsDatabase Classes

Each class in the two taxonomies represented in the ontology is assigned properties based on the previously indicated assignment logic. The exception is the Maintenance activity. Since the Maintenance activity could conceivably include all the other lifecycle activities, properties were not assigned to this class individually. If a person is working in the maintenance phase, they are likely doing a smaller version of one of the other activities and the search should then be based on that specific activity. The alternative would be to assign a relation to Maintenance every time a relationship with a lifecycle activity is made. This would not add anything to the ontology and was therefore left out.

C. Objective Architecture Taxonomy

This taxonomy represents the decomposition of the common architecture for Navy combat systems. The use of the taxonomy will be to enable the software repository to correlate artifacts that have similar relationships based on commonality within the architecture to suggest them as possible items for retrieval. The types of questions for which the information in the taxonomy should provide answers include:

- If I am looking at an artifact related to a particular architectural component, what other artifacts are also related to the same component?

1. Classes and class hierarchy

The Surface Combat System Top-Level Objective Architecture is shown in Figure 56. We constructed a taxonomy from this view of the architecture by declaring each component to be a class and identifying the class hierarchy from the containment indicated in the diagram. Components in the center block of the diagram are part of particular domains (e.g., Display Domain) and also part of the Combat Management Software. This is represented in the taxonomy by assigning two parent classes to a component. For example, the Common GUIs component is a subclass of the Display Domain class and a subclass of the Combat Management Software class. A view of the resulting taxonomy is shown in Figure 57.

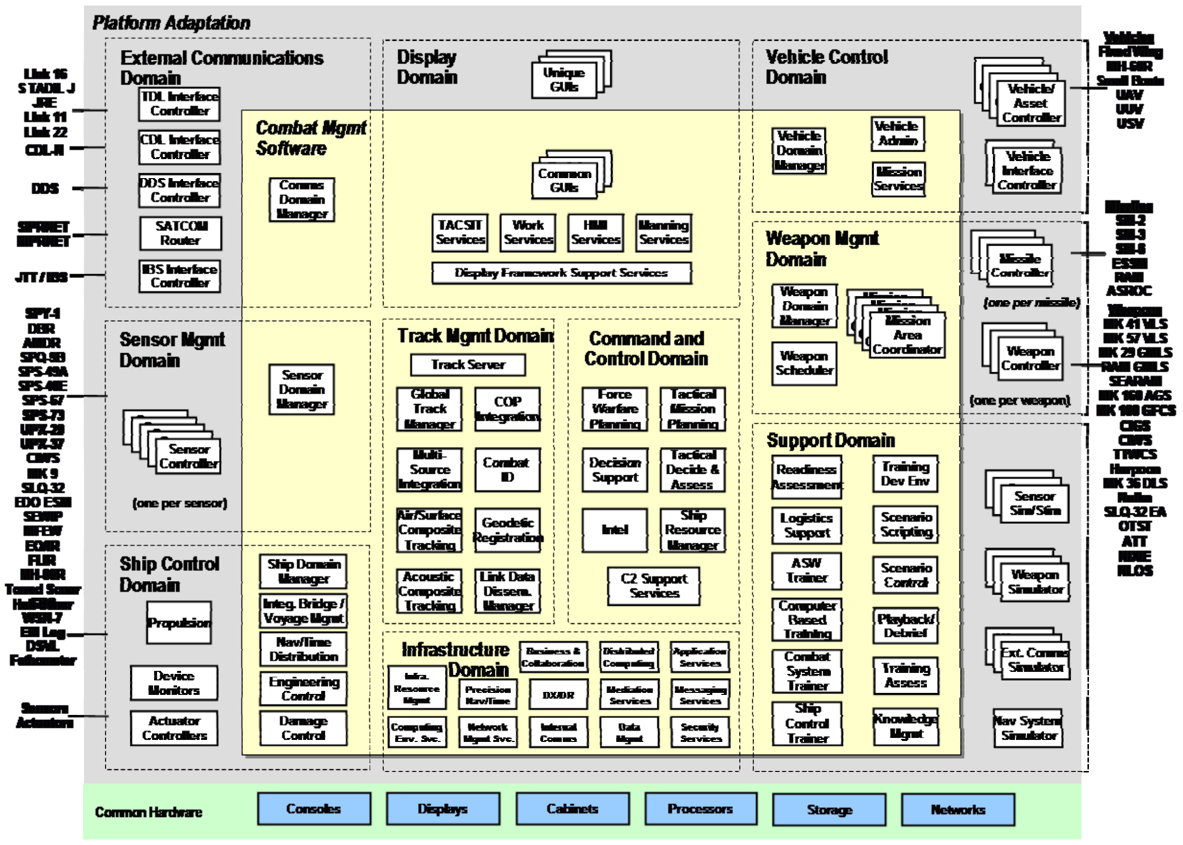
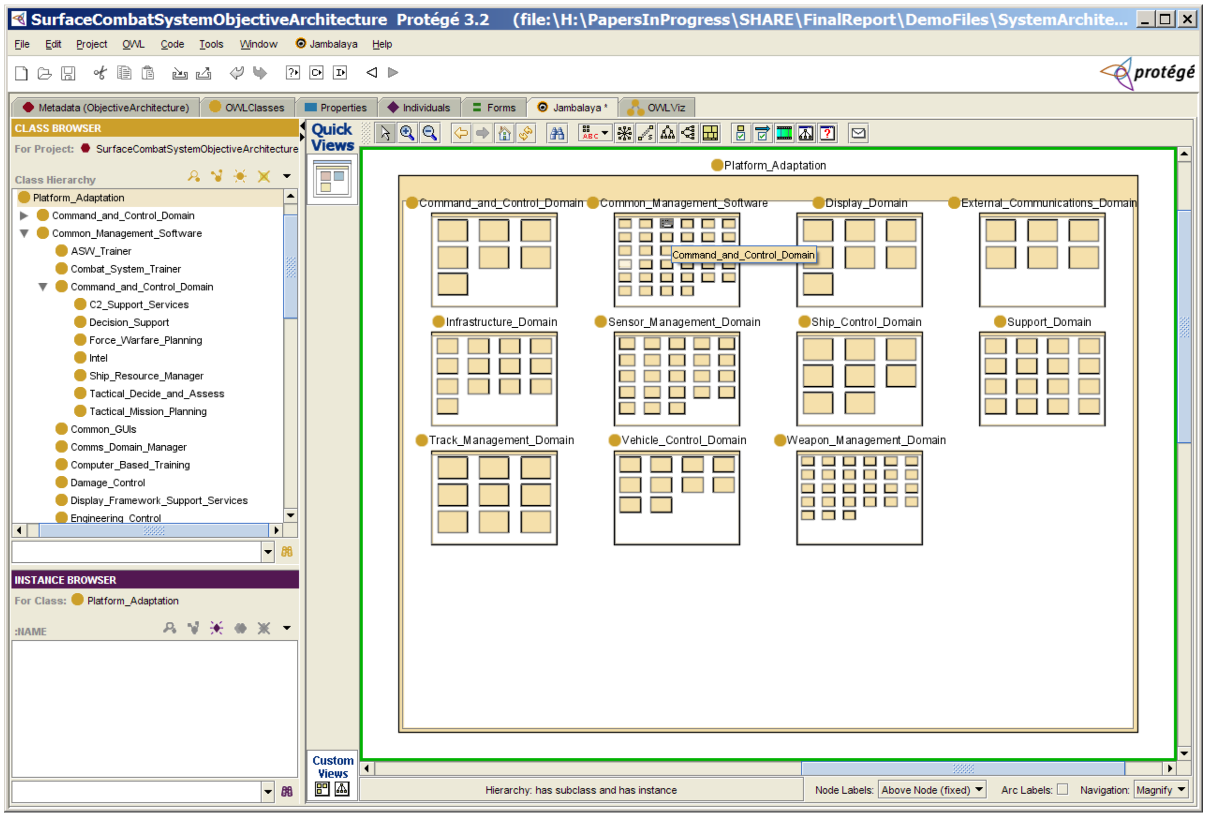


Figure 56. Surface Combat System Top Level Objective Architecture


SurfaceCombatSystemObjectiveArchitecture Protégé 3.2 (file:\H:\PapersInProgress\SHARE\FinalReport\DemoFiles\SystemArchite...

For Project: SurfaceCombatSystemObjectiveArchitecture

For Class: Platform_Adaptation

Hierarchy: has subclass and has instance

Node Labels: Above Node (fixed)

Arc Labels:

Navigation: Magnify

Figure 57. Surface Combat System Top Level Objective Architecture Described as a Taxonomy in OWL (Jambalaya Graphic Tab in Protégé)

In some cases, an entire subclass is contained within a parent class. This is true of the Track Management Domain, Command and Control Domain, and Infrastructure Domain; i.e., every subclass (component) of each of these domains is also a subclass of the Combat Management Software class. In these cases, the “leaf” node subclasses (the components at the bottom of the taxonomy tree, such as Infrastructure Resource Management and Precision Nav/Time from the Infrastructure Domain), are declared in the taxonomy as subclasses of the parent domain, which is declared as a subclass of the overall Platform Adaptation class. Cases in which a domain is not fully contained by the Combat Management Software class, the class membership is declared individually for each component. For example, in the External Communications Domain, the TDL Interface Controller is declared as a subclass of External Communications Domain, while its sibling

Comms Domain Manager component is declared as a subclass of both the External Communications Domain and the Combat Management Software class.

When the asserted Objective Architecture taxonomy is processed through a reasoner (to check for consistency or “classify the taxonomy”), the Track Management Domain, Command and Control Domain, and Infrastructure Domain subclasses are properly moved into the Combat Management Software class. This is correct containment, even though it obscures the original notion that these subclasses can be thought of as “peer” or “sibling” classes to the other domains (e.g., External Communications Domain, Sensor Mangement Domain, Display Domain, etc.). Such issues can be much more subtle in more complex taxonomies and ontologies, illustrating the value in being able to specify precise definitions of concepts and relationships that software can interpret and reason over.

The taxonomy expresses the Common Hardware portion of the objective architecture as a peer class to the Platform Adaptation class. It is envisioned that an individual product (perhaps a hardware configuration item in a specific installation) would be categorized as a Console, Display, Cabinet, Processor, Storage, or Network and described as an individual (a member of the identified subclass) in the knowledge base structured from the taxonomy.

2. Class Properties

Missing from this characterization of the Surface Combat System Objective Architecture is specification of characteristics (properties) of the various subclasses making up the architecture. That is, which characteristics of a component/artifact enable it to be classified (categorized) as a Track Server or, more generally, as a component in the Combat Management Software. What distinguishes a subclass of the Command and Control Domain from a subclass in, for example, the Display Domain? One distinguishing aspect may be obtained through association of functions from the CSFL with subclasses in the Objective Architecture taxonomy. Development of such associations and subclass characterizations is beyond the scope of this effort. However, pursuit of these classification schemes in follow-on work will create greater opportunities for automated reasoning to help identify artifacts within the architecture that can help meet users' needs.

D. System/Subsystem Ontologies

Our recommendation is that ontologies be developed to capture systems, subsystems, and interfaces for each program contained in SHARE. As mentioned in section III.C.1, the system/subsystem taxonomies would be used to verify the entries for the *System* and *Subsystem* elements in the metadata in order to assign artifacts to classes and subclasses (as individuals) within the ontology. Once these are assigned, the repository application could derive interface and other relationships from the ontology.

Concerns about classification issues and our inability to acquire detailed system information limited the depth of this portion of the ontology development effort for the present. Here we provide one example of how systems/subsystems and their interfaces can be captured as an ontology to complement the repository framework.

1. System Ontology Example - Aegis

This ontology will cover the domain of the Aegis combat system baseline and its interfaces. The types of questions for which the information in the ontology should provide answers include:

- What are the system/subsystem relationships for the Aegis baseline?
- What are the interfaces for Aegis, both within the combat system as well as externally?

2. Classes and class hierarchy

This ontology was built based on a portion of a DoD Architecture Framework (DoDAF) system view (SV-1) retrieved from the RDA CHENG Naval Architecture

Repository System (NARS) and shown in Figure 58. The classes in the ontology are the systems/subsystems of the Aegis baseline as well as the systems/subsystems with which they interact within a platform. The current taxonomy provided is at a very high level, and should eventually be complemented with lower levels of granularity to achieve maximum utility.

3. Class Properties

Aside from the "has subclass" relationship that exists between systems and subsystems, there are two properties identified for this ontology.

1. sendsTo—For each system/subsystem, identifies which systems/subsystems to which the information or messages are sent. This is the inverse property of receivesFrom.
2. receivesFrom—For each system/subsystem, identifies which systems/subsystems from which information or messages are received. This is the inverse property of sendsTo.

It may desirable at some point to create subclasses of these properties to specify the information being sent. For example, the sendsTo property could have the sub-property sendsTo_TrackData so that more specific types of interfaces can be queried.

The ontology is depicted in Figure 59. It is actually very similar to the SV-1 depiction, but now exists as an OWL ontology that can be easily referenced in the SHARE metadata and by software; it will interoperate with the other types of ontologies built into the repository framework.

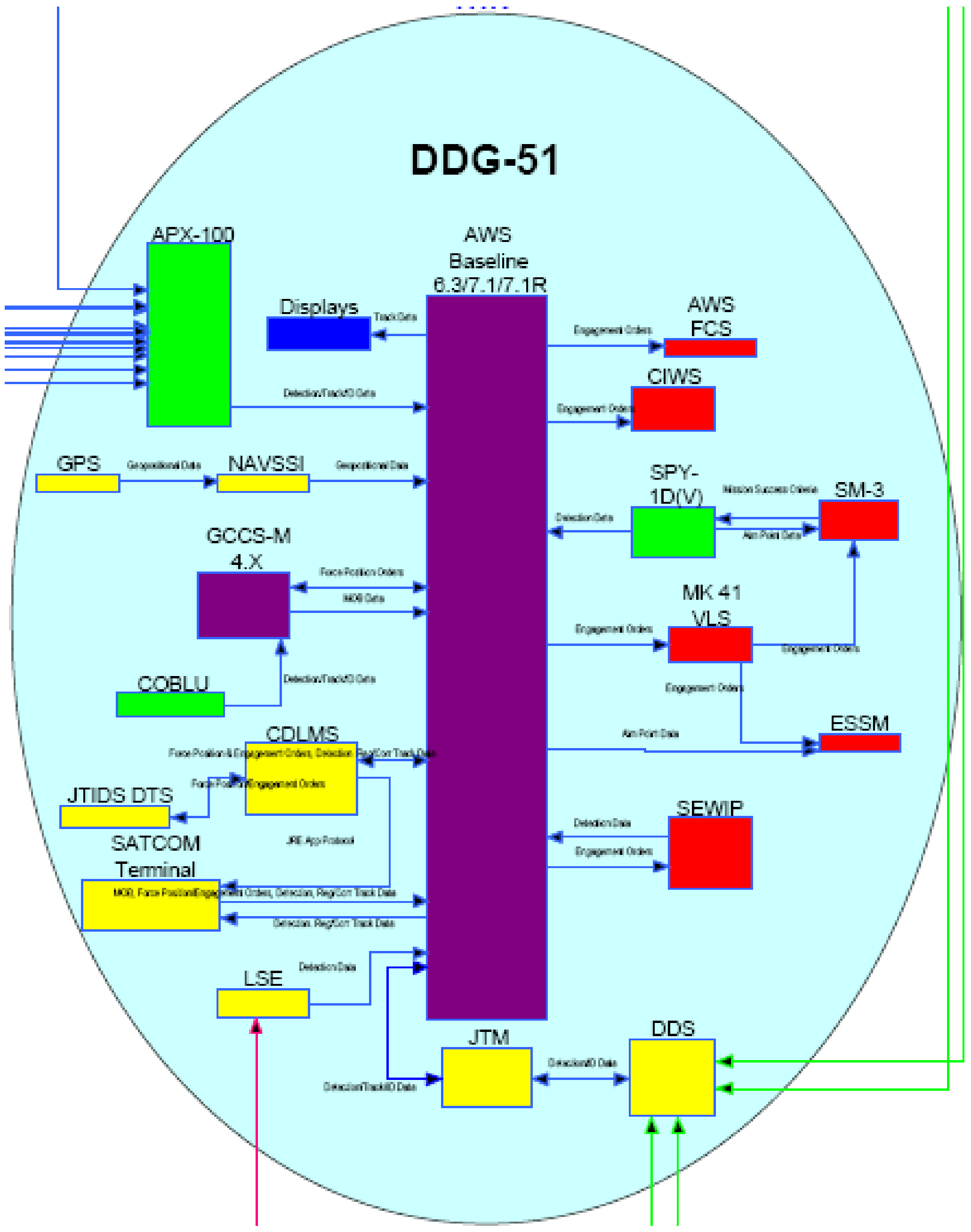


Figure 58. Aegis Combat System SV-1

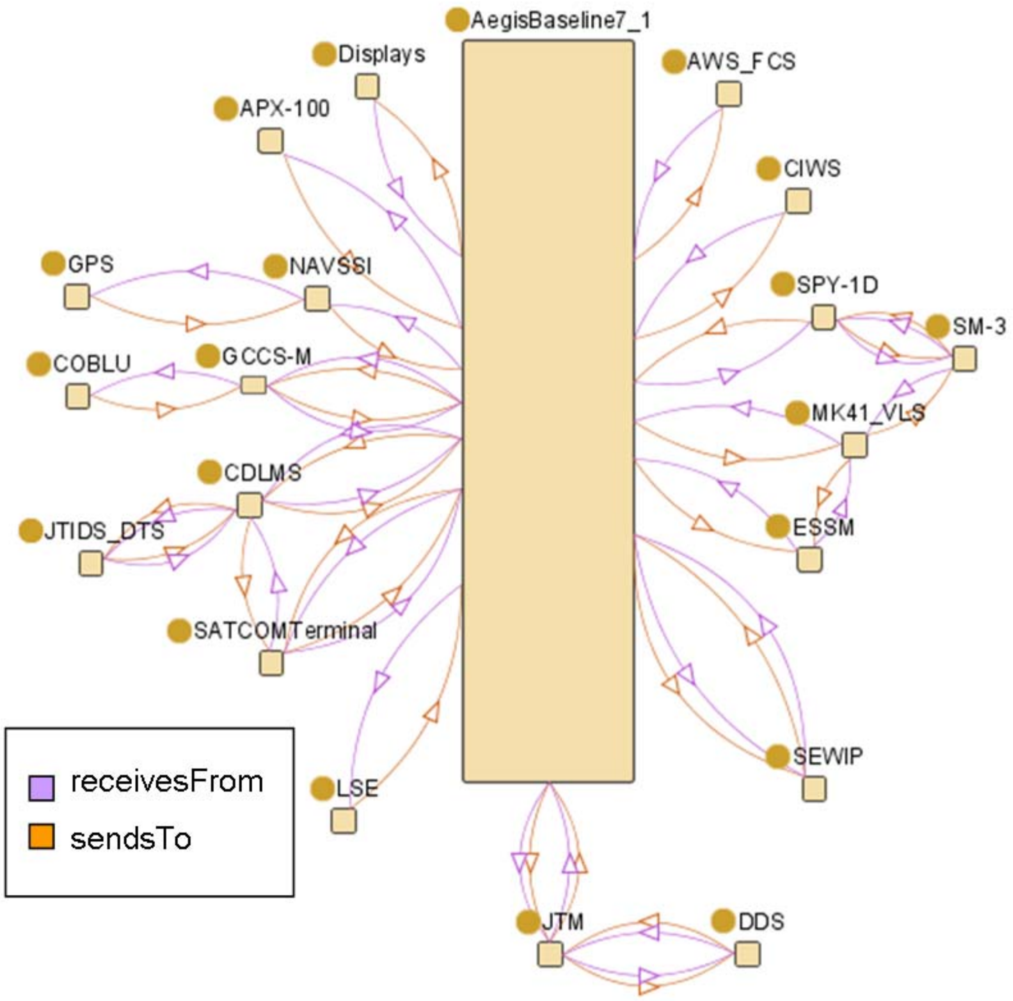


Figure 59. System Ontology Example (Aegis)

It should be noted that the arrows here do not carry the same meaning as those in the SV-1 diagram. The arrows indicate the direction of the property or relationship, not the flow of information. If the AegisBaseline7_1 sendsTo AWS_FCS, as indicated by the orange arrow, then AWS_FCS also receivesFrom AegisBaseline7_1, as indicated by the purple arrow—since these are inverse properties. However, this does not necessarily mean that AWS_FCS also sendsTo AegisBaseline7_1. Bi-directional information flow from the SV-1 is depicted by two sets of bidirectional arrows in the ontology (e.g., between AegisBaseline7_1 and SEWIP).

A great deal more can be done with system ontologies. This example is provided to stimulate thought about what should be included in the system representations and how they can be used within the repository framework.

E. Artifacts as Individuals in the Knowledge Base

The SHARE Relationships Framework described in this section provides a number of conceptual and organizational views that can be used to characterize SHARE assets and artifacts. Another perspective is to consider the taxonomies/ontologies as portions of specifications describing the resources in the SHARE knowledge base. Assets and artifacts would become individuals within that knowledge base characterized by the taxonomy/ontology specifications. That is, an artifact can be identified as a member of a particular class in the framework by virtue of its characteristic properties. The taxonomies/ontologies would then provide associations and relationships between that artifact and others in the knowledge base. Much more detail would need to be added to the currently defined structures, but the foundation is established. This is a potentially important area for future work.

F. Summary

Enriched semantic specification of the assets in the SHARE repository will enable users to more readily find resources that meet their need in their context. Extensive work in the Web community is providing tools and techniques that can be applied to the SHARE framework. We have created an initial semantic foundation on which enhanced capabilities can be implemented.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. Satisfying User Goals

A. Introduction

In this section we investigate two possible scenarios of repository use in order to show how the recommended repository framework will support satisfaction of user goals at the time of search. Two lifecycle phases will be considered for this demonstration—requirements and design.

B. Requirements Phase Scenario

In section II.C, we described potential repository user goals by lifecycle activity. The goals listed for the requirements phase include:

- Users seek to reuse existing requirements, where the proposed system meets existing capabilities.
- Users seek to generate a draft requirements specification from existing repository contents.

In this scenario, imagine that users need to build a replacement for a particular subsystem of Aegis. They consult the SHARE repository to find artifacts that will help in the development of the requirements for the new subsystem. Potentially, there are requirements for an existing system that can be reused. There may also be additional artifacts to be discovered that may be helpful in the requirements development process.

The logical place to begin is by conducting a search for the requirements specifications of the subsystem being replaced. Since our metadata includes a classification of the system/subsystem that each artifact is tied to, a quick sort would reveal applicable artifacts. Narrowing the search to requirements-related artifacts is simple as well, since the metadata also characterizes artifact by type.

For this first step in the search and discovery process, a similar approach could be taken based on the current metadata schema. However, there are two potential improvements. First, since we are able to perform discovery at the artifact level, we eliminate the potential retrieval of unnecessary items. We are also able to view metadata information at a more detailed level, which should help in our evaluation of whether a potentially useful artifact should be retrieved. The second potential improvement provided by the proposed metadata is the approach to the assignment of subsystems, which should be at the lowest level of granularity covered by an artifact. In the current schema, artifacts are grouped into very large subsystems as assets, in many cases making it impossible for the user to determine which available artifacts are specific to the subsystem of interest before retrieving the asset. By assigning the subsystems at lower levels of granularity (and using a system ontology to standardize this taxonomy), users can isolate the search more effectively.

Based on this initial search process, assume that users have identified the available requirements documents for the subsystem to be replaced. The user interface to the repository could provide a bin in which useful identified artifacts can be stored while users continue the search. With these safely set aside, what other artifacts may be helpful?

Since we included a representation of the software behavior tied to artifacts in our metadata, users could potentially identify useful artifacts for subsystems with similar functionality to that of the subsystem being replaced. Imagine an interface that allows users to choose items from the CSFL and then search for items in the repository that also address the same or a similar list of functionality. The returned items could then be further narrowed to artifact types (requirements artifacts in this case) and the detailed metadata consulted if there is a question about whether a returned artifact is going to be useful. In our scenario, users might find the requirements specification for the SSDS subsystem that addresses similar functionality to the one being developed. This item can also be flagged and placed in our bin of items to be retrieved.

In addition to these artifacts discovered through the use of the metadata and software behavior descriptions, potentially, there are helpful additional artifacts that users may not initially consider. The ontologies of the repository framework are key to discovering these artifacts. For example, in the requirements phase, the ontology points to simulation artifacts and white papers as potentially useful types of artifacts. In our scenario, imagine that users are able to navigate through a visual depiction of the ontology to these related items and find a white paper about the subsystem being replaced. Further consultation of the metadata reveals that the paper contains background and explanations regarding decisions made during the requirements analysis. This item is then flagged and added to the bin of items to be retrieved.

The remaining ontologies could be utilized in a similar manner. The objective architecture ontology could be used to identify Aegis or other systems' artifacts tied to the same or closely related subdomains as identified in the objective architecture, and the systems ontology would reveal subsystems that interface with the one being replaced. Artifacts for these subsystems may be needed in order to ensure that the requirements cover the appropriate interoperability issues. For our scenario, users identify interface specifications for related Aegis subsystems that should be consulted and adds them to the retrieval bin.

Once the list of desired artifacts is complete, users can then characterize the package as an asset. The repository tool can walk users through the development of any necessary asset metadata for inclusion in the repository for future users. It can also develop the necessary retrieval process and provide the information to users. For the new metadata schema, this process may be tailored somewhat since multiple PM signatures may be required.

C. Design Phase Scenario

The potential repository user goals we considered for the design phase of the lifecycle identified in section II.C include:

- Users seek to search for existing components that may satisfy portions of the requirements.
- Users seek to retrieve design patterns for common problems.
- Users seek to reason about a system's architecture including the ability to compare and evaluate possible solution compositions, investigate an architecture's ability to satisfy quality attributes or non-functional requirements (interoperability, safety, performance, etc), and investigate an architecture's ability to satisfy functional requirements.

For this scenario, imagine users have set requirements for a subsystem to be built. Users seek artifacts that will aid in the design process.

In this scenario, an appropriate place to start may be to identify the artifacts in the repository related to subsystems with similar functionality to the system being built. To do this, users select from the CSFL listings functionality that relates to the functional requirements specificied for the new system. Artifacts that have been related to similar CSFL groupings are identified by the repository tool.

These artifacts can then be filtered by various characteristics to focus on desired items for retrieval. Artifacts not related to the design phase may be eliminated. Metadata is examined to evaluate if potential items are of interest, desirable, and so on. The result is a group of artifacts that users wish to retrieve, along with the required retrieval information.

While utilizing the same underlying framework, this example demonstrates that the search process may be different depending on user goals. In the first scenario, users quickly determine an individual item of interest and expand the list of retrieved items using the linkages provided by the ontologies. In this scenario, users quickly identify a group of potential items of interest and then use the framework to focus on the desired items.

This scenario also illustrates the premise for the search engine being created as part of the ReSEARCH project. In the future it may be possible to input existing requirements documents into the ReSEARCH engine and automatically derive artifacts of interest. The ReSEARCH engine will likely utilize data from the repository framework, and the framework will benefit from a front end data entry and extraction tool. Further work to integrate these two efforts will likely be required when both groups have concluded their research.

D. Summary

Each piece of the repository framework enhances the search capabilities in different ways. The basic metadata in the XML schemas provide search criteria for finding components of interest in the repository as well as specific information about the artifacts to determine if they are appropriate for retrieval. OWL taxonomies and ontologies enable identification of functionality and associated resources that may be beneficial to users. In short:

- The metadata is evaluated to enable retrieval decisions.
- The software behavior representations enable searches based on functionality.
- The ontologies point the user to helpful artifacts that they may not have initially considered.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. Recommendations and Future Work

A. Introduction

The clear next step in development of the SHARE repository framework is deciding on a strategy to incorporate the products and approaches from the present work into the techniques and practices of the SHARE repository. While the current work provides a starting point, further work will be necessary to implement the framework and develop a tool suite that will enable the described search capabilities. In the SHARE implementation, additional repository features can be added, such as an Amazon-like “similar results” feature that points people with similar problems to the retrieval of the same files and other similar recommendations found in Johnson (2008). Several other recommendations for enriching the semantics of SHARE contents are discussed in the following subsections.

B. SHARE Metadata

The XML schemas developed in this task provide starting points for creating more structured descriptions of SHARE resources. The schemas need to be evaluated for integration into the SHARE software suite. The SHAREAssetInformation schema provides the lowest level of entry into describing the assets in XML since it closely matches the current wizard-directed user entry flow. Even with this approach, the schema and entry process can be evaluated to determine if greater precision and regularity can be enabled through stronger restrictions in data values (e.g., phone, date formats and content). Content can be specified and enforced through the XML schema, ultimately simplifying software logic for parsing and data processing.

A major decision will be needed regarding the architectural change from an asset-centric perspective on the data entry to an artifact-centric perspective using the SHAREAssets and SHAREArtifacts schemas. In the long run, we believe entry and control of information at this level will provide great benefits, but recognize that it would require significant change to the current approach. It may be possible to devise approaches for incremental improvement in this area to reduce impact on ongoing repository use.

The report described some conditions that are difficult to specify in the XML Schema language, such as certain business or usage rules (e.g., given a certain entry, other entries follow). It will be valuable to investigate other schema languages (e.g., Schematron) or formal expressions (e.g., ontologies or rule sets) that can help address such conditions.

A significant challenge will be generation of XML metadata from existing SHARE resources and helping users describe future submissions to the repository. Current research into automatic generation of metadata from content libraries should be explored for potential application to the SHARE context.

C. SHARE Software Behavior Representation

Providing a practical software behavior representation remains a challenging area for continued exploration. The current work provides an identification of principal functionality of an artifact through the CSFL and possible description of operations and input/output messages from a service perspective using WSDL. The work is admittedly preliminary. If the transition is made to more detailed specification of artifacts making up an asset, then more detailed specification of software behavior needs to be investigated. Research into related areas of Semantic Web Services, Business Process Execution Language, and others continues to hold promise for this aspect of the SHARE repository framework.

In addition to the CSFL, similar lists have been developed for operational activities (COAL) and for information elements (CIEL). It would be interesting to express these taxonomies in OWL, as was done with CSFL, and then to create interrelationships across the classes, for example, to determine what information elements are generally employed in performing certain system functions, or what information elements are generally produced by performing certain system functions. Further exploration with subject matter experts (SMEs) is needed to determine potential benefit from such approaches.

As discussed in the report, determining the appropriate level of decomposition of software components, with associated descriptions, needs additional study. Use cases vetted through the SHARE designers and user community will help resolve this issue. Finally, in the long term, further work will also be required if the intent is to eventually enable automated composition of a system based on reusable components. As discussed previously, a starting point to accomplishing this goal may be to standardize a formal behavior representation of the repository contents.

D. SHARE Relationship Framework (Ontology)

The SHARE information entry process requires manual idenfication of relationships across assets (and potentially, artifacts). As the metadata structures become employed and as development practices improve to provide greater description of development artifacts, it will become more practical to derive relationships across the resources in the repository and resources being added to the repository. Such techniques will become increasingly important and efficient as the repository semantics evolve. We recommend continued work to explore and evaluate approaches for automating assignment and derivation of relationships across assets and artifacts in the repository.

As with any knowledge representation approach, proposed representations (i.e., taxonomies, ontologies, rules, logic) need to be vetted by SMEs prior to implementation. As mentioned above, establishment of an appropriate set of use cases will assist in this process. The vetting process will result in refined representations. Additional required ontologies will likely be identified as well, such as the appropriate system ontologies, as discussed in section V.

As discussed in the report, the characterization of the Surface Combat System Objective Architecture is missing a specification of characteristics (properties) of the various subclasses making up the architecture. That is, the characteristics of a component/artifact that enable it to be classified (categorized) as a Track Server or, more generally, as a component in the Combat Management Software. Development of such associations and subclass characterizations will create greater opportunities for automated reasoning to help identify artifacts within the architecture that can help meet users' needs. Further work is also needed across the taxonomies and ontologies to determine what will constitute individuals in the knowledge base (i.e., the totality of class structures and individual members of the classes). It will be valuable to consider an Asset or an Artifact as an individual in the knowledge base, whereby associations and relations can be automatically derived by software reasoners. It is not clear, however, how the characterizations would be defined. More detail needs to be added to the currently defined structures, but the foundation is established for future enhancement.

E. Related Work

Products from the current work need to be integrated with the NPS ReSEARCH project. The SHARE repository framework taxonomies and ontologies provide vocabulary and context specific to SHARE repository products that are expected to improve efficiency and accuracy in searching for contextually-relevant information.

VIII. Summary

This research effort described a component specification and ontology for a SHARE repository framework. The following products were developed from this work (delivered on CD to the program office):

- XML Schema aligned with data entry wizard (SHAREAssetInformation.xsd)
- XML Schemas for Assets and Artifacts with ontology references (SHAREAssets.xsd and SHAREArtifacts.xsd, respectively)
- XSLT for transforming CSFL data (XML exported from Microsoft Excel spreadsheet) to OWL (CSFLspreadsheetToOWL.xslt)
- CSFL taxonomy in OWL and Protégé project file, generated by the XSLT above (CSFL.owl and CSFL.pprj)
- Surface Combat System Objective Architecture taxonomy in OWL and Protégé project file
- Lifecycle-Artifacts ontology in OWL (Artifacts.owl and Artifacts.pprj)
- Aegis ontology in OWL and Protégé project file (Aegis.owl and Aegis.pprj)
- Final Report describing schemas, ontologies, user goals (scenarios), and description of the techniques to meet user goals

A key next step is development of a strategy for beginning to incorporate these products and approaches into the techniques and practices of the SHARE repository.

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

Arora, G., & Kishore, S. (2002). XML web services: Professional projects. Cincinnati: Premier Press.

Computer Human Integration and Software Engineering Lab (CHISEL). (2008). *The CHISEL group, University of Victoria.* Retrieved April 25, 2008, from http://www.thechiselgroup.org/jambalaya

Comprehensive PERL Archive Network (CPAN). (2007). CPAN. Retrieved January 16, 2007, from http://www.cpan.org

Department of Defense Chief Information Officer. (2003, May 9). Net-centric data sharing strategy. Washington, DC.

Deputy Assistant Secretary of Defense. (2007). Department of Defense Discovery Metadata Specification (DDMS) (Version 1.4.1). Washington, DC: Deputy Chief Information Office.

Erl, T. (2005). Service-oriented architecture: Concepts, technology, and design. Upper Saddle River: Pearson Education, Inc.

Johnson, J. (2007, October). SHARE repository component specification: Needs assessment. Technical Report, Monterey, CA: Naval Postgraduate School.

Johnson, J., & Blais, C. (2008, March). Software hardware asset reuse enterprise (SHARE) framework: Related work and development plan. Technical Report. Monterey, CA: Naval Postgraduate School.

Object Management Group. (2005). *Reusable asset specification, Version 2.2.* Retrieved January 29, 2008, from http://www.omg.org/technology/documents/formal/ras.htm

Stanford (2008). *The protégé ontology editor and knowledge acquisition system.* Retrieved April 3, 2008, from http://protege.stanford.edu/

Simventions. (2008, January 22). Modeling and simulation (M&S) community of interest (COI) discovery metadata specification (MSC-DMS) (Version 1.0.1). Washington, DC: DoD Modeling and Simulation Coordination Office (M&S CO).

SourceForge. (2007). *SourceForge.net: Welcome to SourceForge.net.* Retrieved October 8, 2007, from www.sourceforge.net

Szyperski, C. (2002). *Component software: Beyond object-oriented programming, (2nd ed.)*. New York: Addison-Wesley.

Appendix A. XML Schema for Current SHARE Asset Information

The following XML schema describes information corresponding to the major asset information entry steps guided by the SHARE data entry wizard. The information in this schema is discussed in section III.B of this report.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2008 rel. 2 sp1 (http://www.altova.com) by Curtis Blais (Naval Postgraduate
School) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.navy.mil/OpenArchitecture/SHARE"
targetNamespace="http://www.navy.mil/OpenArchitecture/SHARE" elementFormDefault="qualified"
attributeFormDefault="unqualified">
	<xs:element name="SHAREAssetInformation">
		<xs:annotation>
			<xs:documentation>This XML Schema describes information entered into the
Software Hardware Asset Reuse Enterprise (SHARE) Asset Information Form.</xs:documentation>
		</xs:annotation>
		<xs:complexType>
			<xs:sequence>
				<xs:element name="SourceIdentification"
type="SourceIdentificationType">
					<xs:annotation>
						<xs:documentation>Identify the source of the
asset.</xs:documentation>
					</xs:annotation>
				</xs:element>
				<xs:element name="ProgramInformation"
type="ProgramInformationType">
					<xs:annotation>
						<xs:documentation>Identify the program that funded the
development of this asset and that will be responsible for approving release of this asset.</xs:documentation>
					</xs:annotation>
				</xs:element>
				<xs:element name="AssetDescription" type="AssetDescriptionType">
					<xs:annotation>
						<xs:documentation>Descriptive information about the
asset, including its type, classification, and rationale for submitting the asset.</xs:documentation>
					</xs:annotation>
				</xs:element>
				<xs:element name="AssetScope" type="AssetScopeType">
					<xs:annotation>
						<xs:documentation>Information about where and how the
asset may be reused. Asset Scope specifies whether the asset was developed for a tactical application, whether it
is used in the development support environment, or whether it supports reuse in some other way. For a selected
scope, identify the Asset Category to help describe how the asset is intended to be reused. Is it a complete
```

```
system or application that can be used as is? Is it a component or library that will be integrated into another
application?</xs:documentation>
                                        </xs:annotation>
                                </xs:element>
                                <xs:element name="RelatedAssets" type="RelatedAssetsType">
                                        <xs:annotation>
                                                <xs:documentation>Identifies other assets in SHARE that
are related to the asset being submitted. This is important if the asset being submitted i a newer version of an
existing asset. If other assets were used to develop it, or if other assets are required to make it work.
</xs:documentation>
                                        </xs:annotation>
                                </xs:element>
                                <xs:element name="DevelopmentStatus" type="DevelopmentStatusType">
                                        <xs:annotation>
                                                <xs:documentation>Collects information about the
development status (still in development, development complete, planned updates, asset
maturity).</xs:documentation>
                                        </xs:annotation>
                                </xs:element>
                                <xs:element name="ContextInformation" type="ContextInformationType">
                                        <xs:annotation>
                                                <xs:documentation>Collects context information about
the asset (if there are any commercial off the shelf or government off the shelf products required to use the
asset, target operating system(s), programming language(s), and runtime environment(s).</xs:documentation>
                                        </xs:annotation>
                                </xs:element>
                                <xs:element name="ArtifactsContained" type="ArtifactsContainedType">
                                        <xs:annotation>
                                                <xs:documentation>Collects information about the
content of the asset. Artifacts are the individual files that will be submitted for the asset.</xs:documentation>
                                        </xs:annotation>
                                </xs:element>
                                <xs:element name="CombatSystemsObjectiveArchitecture"
type="CombatSystemsObjectiveArchitectureType">
                                        <xs:annotation>
                                                <xs:documentation>Identify where the asset fits into the
Surface Navy Combat Systems Objective Architecture.</xs:documentation>
                                        </xs:annotation>
                                </xs:element>
                                <xs:element name="DataFormat" type="DataFormatType">
                                        <xs:annotation>
                                                <xs:documentation>Provide format and size information
about the physical media and files that will be submitted for the asset. This information is used in the metrics
reported for SHARE and also by the SHARE team to plan for evaluating and uploading the new
asset.</xs:documentation>
                                        </xs:annotation>
                                </xs:element>
                                <xs:element name="RightsAndRestrictions"
type="RightsAndRestrictionsType">
                                        <xs:annotation>
                                                <xs:documentation>List any data rights marking
contained within the asset and list any other licenses that must be obtained to use the asset.</xs:documentation>
                                        </xs:annotation>
```

```
                    </xs:element>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <!--++++++++++++++++++++++++-->
        <!-- Complex Types (alphabetical) -->
        <!--++++++++++++++++++++++++-->
        <xs:complexType name="AddressType">
            <xs:annotation>
                <xs:documentation>Mailing (postal) address.</xs:documentation>
            </xs:annotation>
            <xs:sequence>
                <xs:element name="StreetAddress" type="StreetAddressType"/>
                <xs:element name="MailingAddress2" type="xs:string" minOccurs="0"/>
                <xs:element name="City" type="CityType"/>
                <xs:element name="State" type="StateType"/>
                <xs:element name="Zip" type="ZipType"/>
            </xs:sequence>
        </xs:complexType>
        <xs:complexType name="ArtifactsContainedType">
            <xs:annotation>
                <xs:documentation>Type of artifact included in the asset. Then for each artifact type
selected, briefly describe what artifacts are included and their formats.</xs:documentation>
            </xs:annotation>
            <xs:choice>
                <xs:element name="DocumentArtifact" type="DocumentArtifactType"/>
                <xs:element name="CodeArtifactType" type="CodeArtifactType"/>
            </xs:choice>
        </xs:complexType>
        <xs:complexType name="ArtifactsType">
            <xs:annotation>
                <xs:documentation>List of artifacts contained in an asset.</xs:documentation>
            </xs:annotation>
            <xs:sequence>
                <xs:element name="Artifact" type="ArtifactType" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
        <xs:complexType name="AssetDescriptionType">
            <xs:annotation>
                <xs:documentation>Description of an asset (name, type, description, version, date,
classification, and rationale).</xs:documentation>
            </xs:annotation>
            <xs:sequence>
                <xs:element name="AssetName" type="AssetNameType"/>
                <xs:element name="TypeAsset" type="TypeAssetType"/>
                <xs:element name="Description" type="DescriptionType"/>
                <xs:element name="Version" type="VersionType">
                    <xs:annotation>
                        <xs:documentation>Version or revision identifier for the
asset.</xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="Date" type="DateType" minOccurs="0">
                    <xs:annotation>
                        <xs:documentation>Date indicating the age of the asset, such as
the release date or build date.</xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="ClassificationInformation"
type="ClassificationInformationType">
                    <xs:annotation>
                        <xs:documentation>Indicates the highest level of classification for
any information in the asset, whether there are export controls on any part of the asset and the distribution
statement.</xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="Rationale" type="RationaleType">
                    <xs:annotation>
                        <xs:documentation>Describe why the asset is being
submitted.</xs:documentation>
                    </xs:annotation>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
        <xs:complexType name="AssetScopeType">
            <xs:annotation>
                <xs:documentation>Identification of the scope of the asset (tactical application,
development support, other, or unknown).</xs:documentation>
            </xs:annotation>
            <xs:choice>
                <xs:element name="TacticalApplication"
type="TacticalApplicationCategoryType"/>
                <xs:element name="DevelopmentSupport"
type="DevelopmentSupportCategoryType"/>
                <xs:element name="OtherScope" type="OtherScopeCategoryType"/>
                <xs:element name="UnknownScope"/>
            </xs:choice>
        </xs:complexType>
        <xs:complexType name="ClassificationInformationType">
            <xs:annotation>
                <xs:documentation>Classification data (classification, export control statement,
distribution statement, and security classification guide).</xs:documentation>
            </xs:annotation>
            <xs:sequence>
                <xs:element name="Classification" type="ClassificationType"/>
                <xs:element name="ExportControlStatement"
type="ExportControlStatementType"/>
                <xs:element name="DistributionStatement" type="DistributionStatementType"/>
                <xs:element name="SecurityClassificationGuideID"
type="SecurityClassificationGuideIDType" minOccurs="0"/>
            </xs:sequence>
        </xs:complexType>
        <xs:complexType name="CodeArtifactType">
            <xs:annotation>
                <xs:documentation>Types of artifacts included in the asset. For each artifact type,
briefly describe what artifacts are included and their formats.</xs:documentation>
```

```
				</xs:annotation>
				<xs:choice maxOccurs="6">
					<xs:element name="CodeTypeArtifacts" type="CodeTypeArtifactsType"/>
				</xs:choice>
			</xs:complexType>
			<xs:complexType name="CodeTypeArtifactsType">
				<xs:annotation>
					<xs:documentation>Description of artifacts that are categorized as code
artifacts.</xs:documentation>
				</xs:annotation>
				<xs:sequence>
					<xs:element name="ArtifactType" type="CodeTypeArtifactType"/>
					<xs:element name="Artifacts" type="ArtifactsType"/>
				</xs:sequence>
			</xs:complexType>
			<xs:complexType name="CombatSystemsObjectiveArchitectureType">
				<xs:annotation>
					<xs:documentation>Identifies one or more domains of the Surface Navy Combat
Systems Objective Architecture that apply to the asset being submitted.</xs:documentation>
				</xs:annotation>
				<xs:sequence>
					<xs:element name="ArchitectureDomain" type="ArchitectureDomainType"
maxOccurs="12"/>
				</xs:sequence>
			</xs:complexType>
			<xs:complexType name="ContractingOfficerType">
				<xs:annotation>
					<xs:documentation>Contracting officer contact information.</xs:documentation>
				</xs:annotation>
				<xs:sequence>
					<xs:element name="Name" type="NameType"/>
					<xs:element name="Organization" type="OrganizationNameType"/>
					<xs:element name="Address" type="xs:string"/>
				</xs:sequence>
			</xs:complexType>
			<xs:complexType name="ContractorOrganizationType">
				<xs:annotation>
					<xs:documentation>Identification of the contractor organizatoin (contract number,
delivery, and contracting officer).</xs:documentation>
				</xs:annotation>
				<xs:sequence>
					<xs:element name="ContractNumber" type="ContractNumberType"/>
					<xs:element name="Delivery" type="DeliveryType"/>
					<xs:element name="ContractingOfficer" type="ContractingOfficerType"/>
				</xs:sequence>
			</xs:complexType>
			<xs:complexType name="ContextInformationType">
				<xs:annotation>
					<xs:documentation>Description of the development and operational context of the
asset (COTS or GOTS dependencies, target operating systems, programming languages, and runtime
environments).</xs:documentation>
				</xs:annotation>
				<xs:sequence>
```

```
            <xs:element name="COTSorGOTSDependencies"
type="COTSorGOTSDependenciesType" minOccurs="0"/>
            <xs:element name="TargetOperatingSystems"
type="TargetOperatingSystemsType"/>
            <xs:element name="ProgrammingLanguages"
type="ProgrammingLanguagesType"/>
            <xs:element name="RuntimeEnvironments" type="RuntimeEnvironmentsType"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="COTSorGOTSDependenciesType">
        <xs:annotation>
            <xs:documentation>List of dependencies on COTS or GOTS
products.</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="COTSorGOTSDependency"
type="COTSorGOTSDependencyType" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="DataFormatType">
        <xs:annotation>
            <xs:documentation>Format and size information about the asset being
submitted.</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="PhysicalMediaFormat" type="PhysicalMediaFormatType"
minOccurs="0"/>
            <xs:element name="NumberOfFiles" type="xs:nonNegativeInteger"
minOccurs="0"/>
            <xs:element name="ArchiveFormats" type="ArchiveFormatsType"
minOccurs="0"/>
            <xs:element name="TotalDataSize" type="xs:nonNegativeInteger"
minOccurs="0"/>
            <xs:element name="KSLOCs" type="xs:nonNegativeInteger" minOccurs="0"/>
            <xs:element name="TotalLinesOfComments" type="xs:nonNegativeInteger"
minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="DevelopmentStatusType">
        <xs:annotation>
            <xs:documentation>Description of the status of the asset (planned updates, and asset
maturity).</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="Status" type="AssetDevelopmentStatusType"/>
            <xs:element name="PlannedUpdates" type="PlannedUpdatesType" minOccurs="0">
                <xs:annotation>
                    <xs:documentation>Describe any planned updates to the
asset.</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element name="AssetMaturity" type="AssetMaturityType" minOccurs="0">
                <xs:annotation>
```

```
                                        <xs:documentation>Describe the maturity of the asset: completed?
deployed? included in a system?</xs:documentation>
                                </xs:annotation>
                        </xs:element>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="DocumentArtifactType">
                <xs:annotation>
                        <xs:documentation>Types of artifacts included in the asset. For each artifact type,
briefly describe what artifacts are included and their formats.</xs:documentation>
                </xs:annotation>
                <xs:choice maxOccurs="6">
                        <xs:element name="DocumentTypeArtifacts"
type="DocumentTypeArtifactsType"/>
                </xs:choice>
        </xs:complexType>
        <xs:complexType name="DocumentTypeArtifactsType">
                <xs:annotation>
                        <xs:documentation>Description of artifacts that are categorized as document
artifacts.</xs:documentation>
                </xs:annotation>
                <xs:sequence>
                        <xs:element name="ArtifactType" type="DocumentTypeArtifactType"/>
                        <xs:element name="Artifacts" type="ArtifactsType"/>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="GovernmentOrganizationType">
                <xs:annotation>
                        <xs:documentation>Identification of patents associated with the government
organization submitting the asset.</xs:documentation>
                </xs:annotation>
                <xs:sequence>
                        <xs:element name="ApplicableGovernmentPatents">
                                <xs:complexType>
                                        <xs:sequence>
                                                <xs:element name="Patent" type="PatentType"
minOccurs="0" maxOccurs="unbounded"/>
                                        </xs:sequence>
                                </xs:complexType>
                        </xs:element>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="MemberProfileType">
                <xs:annotation>
                        <xs:documentation>Contact information on the SHARE member submitting the
asset.</xs:documentation>
                </xs:annotation>
                <xs:sequence>
                        <xs:element name="Name" type="NameType"/>
                        <xs:element name="Organization" type="OrganizationNameType"/>
                        <xs:element name="Address" type="AddressType"/>
                        <xs:element name="Phone" type="PhoneType"/>
                        <xs:element name="Email" type="EmailType"/>
```

```
					<xs:element name="Fax" type="PhoneType" minOccurs="0"/>
					<xs:element name="CellPhone" type="PhoneType" minOccurs="0"/>
					<xs:element name="PagerNumber" type="PhoneType" minOccurs="0"/>
					<xs:element name="Notification" type="NotificationType" minOccurs="0"/>
					<xs:element name="ExpirationDates" type="xs:string" minOccurs="0"/>
					<xs:element name="Groups" type="xs:string" minOccurs="0"/>
				</xs:sequence>
		</xs:complexType>
		<xs:complexType name="NotificationType">
				<xs:annotation>
					<xs:documentation>Identify what to notify the member of (baseline design review or
SCRIBE).</xs:documentation>
				</xs:annotation>
				<xs:sequence>
					<xs:element name="BaselineDesignReview" type="xs:boolean"/>
					<xs:element name="SCRIBE" type="xs:boolean"/>
				</xs:sequence>
		</xs:complexType>
		<xs:complexType name="OrganizationType">
				<xs:annotation>
					<xs:documentation>Name and type of organization submitting the
asset.</xs:documentation>
				</xs:annotation>
				<xs:sequence>
					<xs:element name="OrganizationSubmittingAsset"
type="OrganizationNameType"/>
					<xs:element name="TypeOrganization" type="TypeOrganizationType"/>
				</xs:sequence>
		</xs:complexType>
		<xs:complexType name="ProgramInformationType">
				<xs:annotation>
					<xs:documentation>Program name, type, and manager associated with the asset
being submitted.</xs:documentation>
				</xs:annotation>
				<xs:sequence>
					<xs:element name="Program" type="ProgramType"/>
					<xs:element name="ProgramName" type="ProgramNameType" minOccurs="0"/>
					<xs:element name="ProgramManager" type="ProgramManagerType"
minOccurs="0"/>
				</xs:sequence>
		</xs:complexType>
		<xs:complexType name="ProgramManagerType">
				<xs:annotation>
					<xs:documentation>Contact information for the program manager who will need to
approve the release of the asset to SHARE and approve Terms of Use Agreements with other organizations who
want to use the asset.</xs:documentation>
				</xs:annotation>
				<xs:sequence>
					<xs:element name="Name" type="NameType"/>
					<xs:element name="Organization" type="OrganizationNameType"/>
					<xs:element name="Address" type="AddressType"/>
					<xs:element name="Phone" type="PhoneType"/>
					<xs:element name="Email" type="EmailType"/>
```

```
			</xs:sequence>
		</xs:complexType>
		<xs:complexType name="ProgrammingLanguagesType">
			<xs:annotation>
				<xs:documentation>List of programming languages relevant to the asset being
submitted.</xs:documentation>
			</xs:annotation>
			<xs:sequence>
				<xs:element name="ProgrammingLanguage" type="ProgrammingLanguageType"
maxOccurs="unbounded"/>
			</xs:sequence>
		</xs:complexType>
		<xs:complexType name="RelatedAssetType">
			<xs:annotation>
				<xs:documentation>Identification of an asset and its relationship to the asset being
submitted.</xs:documentation>
			</xs:annotation>
			<xs:sequence>
				<xs:element name="Asset" type="AssetNameType"/>
				<xs:choice>
					<xs:element name="CreationRelationship"
type="CreationRelationshipType">
						<xs:annotation>
							<xs:documentation>Indicates if the asset was created
from other assets in SHARE.</xs:documentation>
						</xs:annotation>
					</xs:element>
					<xs:element name="WorkingRelationship"
type="WorkingRelationshipType">
						<xs:annotation>
							<xs:documentation>This indicates if the asset works in
conjunction with other assets in SHARE.</xs:documentation>
						</xs:annotation>
					</xs:element>
					<xs:element name="GenerationRelationship"
type="GenerationRelationshipType">
						<xs:annotation>
							<xs:documentation>This indicates if other assets in
SHARE were created from this asset.</xs:documentation>
						</xs:annotation>
					</xs:element>
					<xs:element name="OtherRelationship" type="OtherRelationshipType">
						<xs:annotation>
							<xs:documentation>This indicates if the asset is related in
other ways with other assets in SHARE.</xs:documentation>
						</xs:annotation>
					</xs:element>
				</xs:choice>
			</xs:sequence>
		</xs:complexType>
		<xs:complexType name="RelatedAssetsType">
			<xs:annotation>
```

```
                <xs:documentation>List of assets related to asset being
submitted.</xs:documentation>
            </xs:annotation>
            <xs:sequence>
                <xs:element name="RelatedAsset" type="RelatedAssetType" minOccurs="0"
maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
        <xs:complexType name="RightsAndRestrictionsIndicatorType">
            <xs:annotation>
                <xs:documentation>Indicates presence ("yes" selection) or absence ("no" selection)
of various rights and restrictions associated with the asset being submitted (data rights markings, commercial
software, special licenses, open source software licenses, data rights assertions, and additional
information).</xs:documentation>
            </xs:annotation>
            <xs:choice>
                <xs:element name="YesSelection" type="YesSelectionType"/>
                <xs:element name="NoSelection">
                    <xs:annotation>
                        <xs:documentation>Element is declared as empty (has no
content).</xs:documentation>
                    </xs:annotation>
                    <xs:complexType/>
                </xs:element>
            </xs:choice>
        </xs:complexType>
        <xs:complexType name="RightsAndRestrictionsType">
            <xs:annotation>
                <xs:documentation>Identification of rights and restrictions associated with the asset
being submitted (data rights markings, commercial software, special licenses, open source software licenses,
data rights assertions, and additional information).</xs:documentation>
            </xs:annotation>
            <xs:sequence>
                <xs:element name="DataRightsMarkings"
type="RightsAndRestrictionsIndicatorType"/>
                <xs:element name="CommercialSoftware"
type="RightsAndRestrictionsIndicatorType"/>
                <xs:element name="SpecialLicenses" type="RightsAndRestrictionsIndicatorType"/>
                <xs:element name="OpenSourceSoftwareLicenses"
type="RightsAndRestrictionsIndicatorType"/>
                <xs:element name="DataRightsAssertions"
type="RightsAndRestrictionsIndicatorType"/>
                <xs:element name="AdditionalInformation" minOccurs="0"/>
            </xs:sequence>
        </xs:complexType>
        <xs:complexType name="RuntimeEnvironmentsType">
            <xs:annotation>
                <xs:documentation>List of runtime environments associated with the asset being
submitted.</xs:documentation>
            </xs:annotation>
            <xs:sequence>
                <xs:element name="RuntimeEnvironment" type="RuntimeEnvironmentType"
maxOccurs="unbounded"/>
```

```
				</xs:sequence>
			</xs:complexType>
			<xs:complexType name="SourceIdentificationType">
				<xs:annotation>
					<xs:documentation>Identification of the source (contributor or organization) of the
information about the asset being submitted.</xs:documentation>
				</xs:annotation>
				<xs:sequence>
					<xs:element name="Contributor" type="MemberProfileType">
						<xs:annotation>
							<xs:documentation>Information about the person and organization
submitting the asset.</xs:documentation>
						</xs:annotation>
					</xs:element>
					<xs:element name="Organization" type="OrganizationType">
						<xs:annotation>
							<xs:documentation>Identify the organization submitting the
asset.</xs:documentation>
						</xs:annotation>
					</xs:element>
				</xs:sequence>
			</xs:complexType>
			<xs:complexType name="TargetOperatingSystemsType">
				<xs:annotation>
					<xs:documentation>List of target operating systems associated with the asset being
submitted.</xs:documentation>
				</xs:annotation>
				<xs:sequence>
					<xs:element name="TargetOperatingSystem" type="TargetOperatingSystemType"
maxOccurs="unbounded"/>
				</xs:sequence>
			</xs:complexType>
			<xs:complexType name="TypeOrganizationType">
				<xs:annotation>
					<xs:documentation>Type of organization (government or contractor).
</xs:documentation>
				</xs:annotation>
				<xs:choice>
					<xs:element name="GovernmentOrganization"
type="GovernmentOrganizationType"/>
					<xs:element name="ContractorOrganization" type="ContractorOrganizationType"/>
				</xs:choice>
			</xs:complexType>
			<!--+++++++++++++++++++++++-->
			<!-- Simple Types (alphabetical) -->
			<!--+++++++++++++++++++++++-->
			<xs:simpleType name="ArchitectureDomainType">
				<xs:annotation>
					<xs:documentation>Possible values for the architecture domain of the asset (where
the asset fits into the Surface Navy Combat Systems Objective Architecture) - Computing Equipment,
Infrastructure, Display, Sensor Management, Track Management, Command and Control (C2), Weapon
Management, Vehicle Control, External Communication (EXCOMM), Common Support, Ship Control, or
Other</xs:documentation>
```

```
				</xs:annotation>
				<xs:restriction base="xs:string">
					<xs:enumeration value="Computing Equipment">
						<xs:annotation>
							<xs:documentation>Provides common computer processors,
displays, storage devices, and networking hardware and enclosures, fielded as the combat system computing
environment.</xs:documentation>
						</xs:annotation>
					</xs:enumeration>
					<xs:enumeration value="Infrastructure">
						<xs:annotation>
							<xs:documentation>Provides computing and network services
such as applicatoin state and mode control, messaging, data recording.</xs:documentation>
						</xs:annotation>
					</xs:enumeration>
					<xs:enumeration value="Display">
						<xs:annotation>
							<xs:documentation>Provides the framework for the generation of
graphical user interfaces as well as common operator displays. Includes services for "signing on" to consoles
and application access management.</xs:documentation>
						</xs:annotation>
					</xs:enumeration>
					<xs:enumeration value="Sensor Management">
						<xs:annotation>
							<xs:documentation>Provides sensor detection, tracking and
support capability.</xs:documentation>
						</xs:annotation>
					</xs:enumeration>
					<xs:enumeration value="Track Management">
						<xs:annotation>
							<xs:documentation>Integrates sensor data from local ship sensor
resources and remote sensors and networks to develop common tactical and operational
pictures.</xs:documentation>
						</xs:annotation>
					</xs:enumeration>
					<xs:enumeration value="Command and Control (C2)">
						<xs:annotation>
							<xs:documentation>Provides support for battle management and
tactical decision making.</xs:documentation>
						</xs:annotation>
					</xs:enumeration>
					<xs:enumeration value="Weapon Management">
						<xs:annotation>
							<xs:documentation>Provides combat system weapon engagement
capabilities.</xs:documentation>
						</xs:annotation>
					</xs:enumeration>
					<xs:enumeration value="Vehicle Control">
						<xs:annotation>
							<xs:documentation>Provides capabilities to ensure safety and asset
management for offboard entities controlled by own ship operators.</xs:documentation>
						</xs:annotation>
					</xs:enumeration>
```

```
						<xs:enumeration value="External Communications (EXCOMM)">
							<xs:annotation>
								<xs:documentation>Provides access to shipboard communications
terminals such as JTIDS, Joint Tactical Terminal (JTT), and IMMARSAT to access SIPRNET, NIPRNET and
other tactical data networks.</xs:documentation>
							</xs:annotation>
						</xs:enumeration>
						<xs:enumeration value="Common Support">
							<xs:annotation>
								<xs:documentation>Provides combat system support services such
as training applications and interface simulation.</xs:documentation>
							</xs:annotation>
						</xs:enumeration>
						<xs:enumeration value="Ship Control">
							<xs:annotation>
								<xs:documentation><![CDATA[Provides services for ship damage
control, propulsion control, and monitoring/management of ship hull, mechanical, and electrical
(HM&E).]]></xs:documentation>
							</xs:annotation>
						</xs:enumeration>
						<xs:enumeration value="Other"/>
					</xs:restriction>
				</xs:simpleType>
				<xs:simpleType name="ArchiveFormatsType">
					<xs:annotation>
						<xs:documentation>Identification of the format(s) of the archived asset content.
Type: string</xs:documentation>
					</xs:annotation>
					<xs:restriction base="xs:string">
						<xs:minLength value="1"/>
					</xs:restriction>
				</xs:simpleType>
				<xs:simpleType name="ArtifactType">
					<xs:annotation>
						<xs:documentation>Describes an artifact included in an asset and its format. Type:
string</xs:documentation>
					</xs:annotation>
					<xs:restriction base="xs:string">
						<xs:minLength value="1"/>
					</xs:restriction>
				</xs:simpleType>
				<xs:simpleType name="AssetNameType">
					<xs:annotation>
						<xs:documentation>Name of the asset being submitted. Type:
string</xs:documentation>
					</xs:annotation>
					<xs:restriction base="xs:string">
						<xs:minLength value="1"/>
					</xs:restriction>
				</xs:simpleType>
				<xs:simpleType name="AssetDevelopmentStatusType">
					<xs:annotation>
					<xs:documentation>Possible values for the development status of the asset being 
submitted - In Development or Development Completed</xs:documentation>
				</xs:annotation>
				<xs:restriction base="xs:string">
					<xs:enumeration value="In Development"/>
					<xs:enumeration value="Development Completed"/>
				</xs:restriction>
			</xs:simpleType>
			<xs:simpleType name="AssetMaturityType">
				<xs:annotation>
					<xs:documentation>Maturity of the asset being submitted. Type: 
string</xs:documentation>
				</xs:annotation>
				<xs:restriction base="xs:string">
					<xs:minLength value="1"/>
				</xs:restriction>
			</xs:simpleType>
			<xs:simpleType name="CityType">
				<xs:annotation>
					<xs:documentation>City. Type: string</xs:documentation>
				</xs:annotation>
				<xs:restriction base="xs:string">
					<xs:minLength value="1"/>
				</xs:restriction>
			</xs:simpleType>
			<xs:simpleType name="ClassificationType">
				<xs:annotation>
					<xs:documentation>Possible values for classification of the asset being submitted - 
U, C, or S</xs:documentation>
				</xs:annotation>
				<xs:restriction base="xs:string">
					<xs:enumeration value="U"/>
					<xs:enumeration value="C"/>
					<xs:enumeration value="S"/>
				</xs:restriction>
			</xs:simpleType>
			<xs:simpleType name="CodeTypeArtifactType">
				<xs:annotation>
					<xs:documentation>Possible values for a code type artifact - Source Code, Compiled 
Libraries, Executable Programs, Data/Support Files, Build Scripts/Instructions, or Tools</xs:documentation>
				</xs:annotation>
				<xs:restriction base="xs:string">
					<xs:enumeration value="Source Code"/>
					<xs:enumeration value="Compiled Libraries"/>
					<xs:enumeration value="Executable Programs"/>
					<xs:enumeration value="Data/Support Files"/>
					<xs:enumeration value="Build Scripts/Instructions"/>
					<xs:enumeration value="Tools"/>
				</xs:restriction>
			</xs:simpleType>
			<xs:simpleType name="COTSorGOTSDependencyType">
				<xs:annotation>
```

```
                    <xs:documentation>Description of the dependency on COTS or GOTS. Type:
string</xs:documentation>
                </xs:annotation>
                <xs:restriction base="xs:string">
                    <xs:minLength value="1"/>
                </xs:restriction>
            </xs:simpleType>
            <xs:simpleType name="ContractNumberType">
                <xs:annotation>
                    <xs:documentation>Contract number. Type: string</xs:documentation>
                </xs:annotation>
                <xs:restriction base="xs:string">
                    <xs:minLength value="1"/>
                </xs:restriction>
            </xs:simpleType>
            <xs:simpleType name="CreationRelationshipType">
                <xs:annotation>
                    <xs:documentation>Possible values for describing the creation relationship between
the asset being submitted and other assets - Newer Version, Variant Of, Extracted From, or Derived
From</xs:documentation>
                </xs:annotation>
                <xs:restriction base="xs:string">
                    <xs:enumeration value="Newer Version">
                        <xs:annotation>
                            <xs:documentation>This indicates that the current asset is an
earlier version of the related asset. It is intended to upgrade or replace the related asset, not be a variant of
it.</xs:documentation>
                        </xs:annotation>
                    </xs:enumeration>
                    <xs:enumeration value="Variant Of">
                        <xs:annotation>
                            <xs:documentation>This indicates that the current asset is a variant
of the related asset to be used in a different target envirnoment or situation.</xs:documentation>
                        </xs:annotation>
                    </xs:enumeration>
                    <xs:enumeration value="Extracted From">
                        <xs:annotation>
                            <xs:documentation>This indicates that the current asset was
extracted from another asset and was possibly repackages to be more reusable. The extracted asset will not be
merged back into the asset it was extracted from.</xs:documentation>
                        </xs:annotation>
                    </xs:enumeration>
                    <xs:enumeration value="Derived From">
                        <xs:annotation>
                            <xs:documentation>This indicates that the current asset was
originally based on the related asset but is sufficiently different to consider it a different asset and not a version
or variant.</xs:documentation>
                        </xs:annotation>
                    </xs:enumeration>
                </xs:restriction>
            </xs:simpleType>
            <xs:simpleType name="DateType">
                <xs:annotation>
```

```
                              <xs:documentation>Date of submission of the asset. Type: 
string</xs:documentation>
                         </xs:annotation>
                         <xs:restriction base="xs:string">
                              <xs:minLength value="1"/>
                         </xs:restriction>
                    </xs:simpleType>
                    <xs:simpleType name="DeliveryType">
                         <xs:annotation>
                              <xs:documentation>Possible values for the type of delivery for an asset - DD250, 
CDRL, or Don't Know</xs:documentation>
                         </xs:annotation>
                         <xs:restriction base="xs:string">
                              <xs:enumeration value="DD250"/>
                              <xs:enumeration value="CDRL"/>
                              <xs:enumeration value="Don't Know"/>
                         </xs:restriction>
                    </xs:simpleType>
                    <xs:simpleType name="DescriptionType">
                         <xs:annotation>
                              <xs:documentation>Description of the asset being submitted. Type: 
string</xs:documentation>
                         </xs:annotation>
                         <xs:restriction base="xs:string">
                              <xs:minLength value="1"/>
                         </xs:restriction>
                    </xs:simpleType>
                    <xs:simpleType name="DevelopmentSupportCategoryType">
                         <xs:annotation>
                              <xs:documentation>The possible values for identifying the asset category in the 
Development Support asset scope - Database/Data Files, Framework, Tools/Utilities, or Test 
Tools/Environments </xs:documentation>
                         </xs:annotation>
                         <xs:restriction base="xs:string">
                              <xs:enumeration value="Database/Data Files">
                                   <xs:annotation>
                                        <xs:documentation>Database or data files that are not 
accompanied by the functional code that uses the data.</xs:documentation>
                                   </xs:annotation>
                              </xs:enumeration>
                              <xs:enumeration value="Framework">
                                   <xs:annotation>
                                        <xs:documentation>A reusable design for a software system (or 
subsystem) expressed as a set of abstract classes and the way their instances collaborate for a specific type of 
software. A framework may include support programs, code libraries, a scripting language, or other software to 
help develop and glue together the different components of a software project. (Wikipedia)</xs:documentation>
                                   </xs:annotation>
                              </xs:enumeration>
                              <xs:enumeration value="Tools/Utilities">
                                   <xs:annotation>
                                        <xs:documentation>Code or application that is intended primarily 
to support development activities rather than to be incorporated into tactical programs.</xs:documentation>
                                   </xs:annotation>
```

```
                </xs:enumeration>
                <xs:enumeration value="Test Tools/Environments"/>
            </xs:restriction>
        </xs:simpleType>
        <xs:simpleType name="DistributionStatementType">
            <xs:annotation>
                <xs:documentation>The possible values for the distribution statement applicable to
the artifact - A, B, C, D, E, F, or X</xs:documentation>
            </xs:annotation>
            <xs:restriction base="xs:string">
                <xs:enumeration value="A">
                    <xs:annotation>
                        <xs:documentation>Approved for public release; distribution is
unlimited.</xs:documentation>
                    </xs:annotation>
                </xs:enumeration>
                <xs:enumeration value="B">
                    <xs:annotation>
                        <xs:documentation>Distribution authorized to U.S. Government
Agencies only (fill in reason) (date of determination). Other requests for this document shall be referred to
(insert controlling DoD office).</xs:documentation>
                    </xs:annotation>
                </xs:enumeration>
                <xs:enumeration value="C">
                    <xs:annotation>
                        <xs:documentation>Distribution authorized to U.S. Government
Agencies and their contractors (fill in reason) (date of determination). Other requests for this document shall be
referred to (insert controlling DoD office).</xs:documentation>
                    </xs:annotation>
                </xs:enumeration>
                <xs:enumeration value="D">
                    <xs:annotation>
                        <xs:documentation>Distribution authorized to the Department of
Defense and U.S. DoD contractors only (fill in reason) (date of determination). Other requests shall be referred
to (insert controlling DoD office).</xs:documentation>
                    </xs:annotation>
                </xs:enumeration>
                <xs:enumeration value="E">
                    <xs:annotation>
                        <xs:documentation>Distribution authorized to DoD Components
only (fill in reason) (date of determination). Other requests for this document shall be referred to (insert
controlling DoD office).</xs:documentation>
                    </xs:annotation>
                </xs:enumeration>
                <xs:enumeration value="F">
                    <xs:annotation>
                        <xs:documentation>Further distribution only as directed by (insert
controlling DoD office) (date of determination) or higher DoD authority.</xs:documentation>
                    </xs:annotation>
                </xs:enumeration>
                <xs:enumeration value="X">
                    <xs:annotation>
                    <xs:documentation>Distribution authorized to U.S. Government
Agencies and private individuals or enterprises eligible to obtain export-controlled technical data in accordance
with reference (c) (date of determination). Controlling DoD office is (insert).</xs:documentation>
                </xs:annotation>
            </xs:enumeration>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="DocumentTypeArtifactType">
        <xs:annotation>
            <xs:documentation>The possible values for the type of artifact in a Document type
asset - Requirements, Design/Architecture Documentation, Test Procedures, User Documentation, Training
Documentation, or Test Records</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="Requirements"/>
            <xs:enumeration value="Design/Architecture Documentation"/>
            <xs:enumeration value="Test Procedures"/>
            <xs:enumeration value="User Documentation"/>
            <xs:enumeration value="Training Documentation"/>
            <xs:enumeration value="Test Records"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="EmailType">
        <xs:annotation>
            <xs:documentation>Email address. Type: string</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:pattern value="([A-Z]|[a-z]|[0-9])+@([A-Z]|[a-z]|[0-9])+\.([A-Z]|[a-z]|[0-
9]){3}"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="ExportControlStatementType">
        <xs:annotation>
            <xs:documentation>The possible values for the Export Control Statement on the
asset being submitted - Yes or No</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="Yes"/>
            <xs:enumeration value="No"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="GenerationRelationshipType">
        <xs:annotation>
            <xs:documentation>Possible values for describing the generation relationship
between the asset being submitted and other assets - Older Version, Extraction Source, or Derivation
Source</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="Older Version">
                <xs:annotation>
                    <xs:documentation>This indicates that the current asset is updated
or replaced by the related asset.</xs:documentation>
                </xs:annotation>
```

```
                                </xs:enumeration>
                                <xs:enumeration value="Extraction Source">
                                        <xs:annotation>
                                                <xs:documentation>This indicates that the current asset was used
to extract the related asset. The extracted asset will not be merged back into the aset it was extracted
from.</xs:documentation>
                                        </xs:annotation>
                                </xs:enumeration>
                                <xs:enumeration value="Derivation Source">
                                        <xs:annotation>
                                                <xs:documentation>This indicates that the related asset was
originally based on the current asset but is sufficiently different to consider it a different asset and not a version
or variant.</xs:documentation>
                                        </xs:annotation>
                                </xs:enumeration>
                        </xs:restriction>
                </xs:simpleType>
                <xs:simpleType name="NameType">
                        <xs:annotation>
                                <xs:documentation>Person name. Type: string</xs:documentation>
                        </xs:annotation>
                        <xs:restriction base="xs:string">
                                <xs:minLength value="1"/>
                        </xs:restriction>
                </xs:simpleType>
                <xs:simpleType name="OrganizationNameType">
                        <xs:annotation>
                                <xs:documentation>Name of an organization. Type: string</xs:documentation>
                        </xs:annotation>
                        <xs:restriction base="xs:string">
                                <xs:minLength value="1"/>
                        </xs:restriction>
                </xs:simpleType>
                <xs:simpleType name="OtherRelationshipType">
                        <xs:annotation>
                                <xs:documentation>Possible values for describing other relationships between the
asset being submitted and other assets - Similar To, Contained Within, or Contains</xs:documentation>
                        </xs:annotation>
                        <xs:restriction base="xs:string">
                                <xs:enumeration value="Similar To">
                                        <xs:annotation>
                                                <xs:documentation>This indicates that the other asset has
characteristics that are similar to the current asset.</xs:documentation>
                                        </xs:annotation>
                                </xs:enumeration>
                                <xs:enumeration value="Contained Within">
                                        <xs:annotation>
                                                <xs:documentation>This indicates that the current asset is part of a
larger asset that is also stored in SHARE. This containment may be physical or by
reference.</xs:documentation>
                                        </xs:annotation>
                                </xs:enumeration>
                                <xs:enumeration value="Contains">
```

```
                                        <xs:annotation>
                                                <xs:documentation>This indicates that the current asset contains
the related asset. This containment may be physical or by reference.</xs:documentation>
                                        </xs:annotation>
                                </xs:enumeration>
                        </xs:restriction>
                </xs:simpleType>
                <xs:simpleType name="OtherScopeCategoryType">
                        <xs:annotation>
                                <xs:documentation>The possible values for identifying the asset category in the
Other asset scope - Enterprise Framework, Data Architecture, or Pattern/Design/Algorithm</xs:documentation>
                        </xs:annotation>
                        <xs:restriction base="xs:string">
                                <xs:enumeration value="Enterprise Framework">
                                        <xs:annotation>
                                                <xs:documentation>Description of a current and/or future structure
and behavior for an organization's processes, information systems, peronnel and organizational sub-units so that
they align with the organization's core goals and strategic direction.</xs:documentation>
                                        </xs:annotation>
                                </xs:enumeration>
                                <xs:enumeration value="Data Architecture">
                                        <xs:annotation>
                                                <xs:documentation>Development and execution of architectures,
policies, practices and procedures that properly manage the full data lifecycle needs of an
enterprise.</xs:documentation>
                                        </xs:annotation>
                                </xs:enumeration>
                                <xs:enumeration value="Pattern/Design/Algorithm">
                                        <xs:annotation>
                                                <xs:documentation>Design guidance that captures information that
may be generally applicable but that is not provided in the context of a specific system.</xs:documentation>
                                        </xs:annotation>
                                </xs:enumeration>
                        </xs:restriction>
                </xs:simpleType>
                <xs:simpleType name="PatentType">
                        <xs:annotation>
                                <xs:documentation>Identification of government patent(s) that apply to the asset
being submitted. Type: string</xs:documentation>
                        </xs:annotation>
                        <xs:restriction base="xs:string">
                                <xs:minLength value="1"/>
                        </xs:restriction>
                </xs:simpleType>
                <xs:simpleType name="PhoneType">
                        <xs:annotation>
                                <xs:documentation>String pattern for a telephone number. Type:
string</xs:documentation>
                        </xs:annotation>
                        <xs:restriction base="xs:string">
                                <xs:pattern value="d{3}(\.|-)d{3}(\.|-)d{4}"/>
                        </xs:restriction>
                </xs:simpleType>
```

```
	<xs:simpleType name="PhysicalMediaFormatType">
		<xs:annotation>
			<xs:documentation>The possible values for identifying the physical media format
for the artifact being provided - CD, DVD</xs:documentation>
		</xs:annotation>
		<xs:restriction base="xs:string">
			<xs:enumeration value="CD"/>
			<xs:enumeration value="DVD"/>
		</xs:restriction>
	</xs:simpleType>
	<xs:simpleType name="PlannedUpdatesType">
		<xs:annotation>
			<xs:documentation>Description of any planned updates that are currently known.
Type: string</xs:documentation>
		</xs:annotation>
		<xs:restriction base="xs:string">
			<xs:minLength value="1"/>
		</xs:restriction>
	</xs:simpleType>
	<xs:simpleType name="ProgrammingLanguageType">
		<xs:annotation>
			<xs:documentation>Identification of a programming language applicable to the asset
being submitted. Type: string</xs:documentation>
		</xs:annotation>
		<xs:restriction base="xs:string">
			<xs:minLength value="1"/>
		</xs:restriction>
	</xs:simpleType>
	<xs:simpleType name="ProgramNameType">
		<xs:annotation>
			<xs:documentation>The name of the program (when not previously entered)
applicable to the asset being submitted. Type: string</xs:documentation>
		</xs:annotation>
		<xs:restriction base="xs:string">
			<xs:minLength value="1"/>
		</xs:restriction>
	</xs:simpleType>
	<xs:simpleType name="ProgramType">
		<xs:annotation>
			<xs:documentation>The possible values for identification of a (previously entered)
program applicable to the asset being submitted - AEGIS, DDG 1000, SSDS, LCS, NSWCDD HSI, SIAP,
SQQ-89, TSTS, ASW, CEP WASP, GeDear, BFTT, Other</xs:documentation>
		</xs:annotation>
		<xs:restriction base="xs:string">
			<xs:enumeration value="AEGIS"/>
			<xs:enumeration value="DDG 1000"/>
			<xs:enumeration value="SSDS"/>
			<xs:enumeration value="LCS"/>
			<xs:enumeration value="NSWCDD HSI"/>
			<xs:enumeration value="SIAP"/>
			<xs:enumeration value="SQQ-89"/>
			<xs:enumeration value="TSTS"/>
			<xs:enumeration value="ASW"/>
```

```
                        <xs:enumeration value="CEP WASP"/>
                        <xs:enumeration value="GeDear"/>
                        <xs:enumeration value="BFTT"/>
                        <xs:enumeration value="Other"/>
                </xs:restriction>
        </xs:simpleType>
        <xs:simpleType name="RationaleType">
                <xs:annotation>
                        <xs:documentation>Description of the rationale for submitting the asset. Type: 
string</xs:documentation>
                </xs:annotation>
                <xs:restriction base="xs:string">
                        <xs:minLength value="1"/>
                </xs:restriction>
        </xs:simpleType>
        <xs:simpleType name="RuntimeEnvironmentType">
                <xs:annotation>
                        <xs:documentation>Identification of a runtime environment applicable to the asset 
being submitted. Type: string</xs:documentation>
                </xs:annotation>
                <xs:restriction base="xs:string">
                        <xs:minLength value="1"/>
                </xs:restriction>
        </xs:simpleType>
        <xs:simpleType name="SecurityClassificationGuideIDType">
                <xs:annotation>
                        <xs:documentation>Identification of the Security Classification Guide ID# for a 
classified asset. Type: string</xs:documentation>
                </xs:annotation>
                <xs:restriction base="xs:string">
                        <xs:minLength value="1"/>
                </xs:restriction>
        </xs:simpleType>
        <xs:simpleType name="StateType">
                <xs:annotation>
                        <xs:documentation>The possible values for the US state in an address - AL, AK, 
AZ, AR, CA, CO, CT, DE, DC, FL, GA, HI, ID, IL, IN, IA, KS, KY, LA, ME, MD, MA, MI, MN, MS, MO, 
MT, NE, NV, NH, NJ, NM, NY, NC, ND, OH, OK, OR, PA, RI, SC, SD, TN, TX, UT, VT, VA, WA, WV, WI, 
or WY</xs:documentation>
                </xs:annotation>
                <xs:restriction base="xs:string">
                        <xs:pattern 
value="AL|AK|AZ|AR|CA|CO|CT|DE|DC|FL|GA|HI|ID|IL|IN|IA|KS|KY|LA|ME|MD|MA|MI|MN|MS|MO|MT|
NE|NV|NH|NJ|NM|NY|NC|ND|OH|OK|OR|PA|RI|SC|SD|TN|TX|UT|VT|VA|WA|WV|WI|WY"/>
                </xs:restriction>
        </xs:simpleType>
        <xs:simpleType name="StreetAddressType">
                <xs:annotation>
                        <xs:documentation>Street address. Type: string</xs:documentation>
                </xs:annotation>
                <xs:restriction base="xs:string">
                        <xs:minLength value="1"/>
                </xs:restriction>
        </xs:simpleType>
        <xs:simpleType name="TacticalApplicationCategoryType">
                <xs:annotation>
                        <xs:documentation>The possible values for identifying the asset category in the
Tactical Application asset scope - System, Application Program, Package, System Service, Component,
Library, or Module/Code Fragment</xs:documentation>
                </xs:annotation>
                <xs:restriction base="xs:string">
                        <xs:enumeration value="System">
                                <xs:annotation>
                                        <xs:documentation>Collection of applicatoins and data that work
together to support a mission.</xs:documentation>
                                </xs:annotation>
                        </xs:enumeration>
                        <xs:enumeration value="Application Program">
                                <xs:annotation>
                                        <xs:documentation>Complete computer program or applet that
will execute in the target environment without being incorporated into a larger program.</xs:documentation>
                                </xs:annotation>
                        </xs:enumeration>
                        <xs:enumeration value="Package">
                                <xs:annotation>
                                        <xs:documentation>A collection of related components of system
services.</xs:documentation>
                                </xs:annotation>
                        </xs:enumeration>
                        <xs:enumeration value="System Service">
                                <xs:annotation>
                                        <xs:documentation>A system element element offering a
predefined service and able to communicate with other components via a well defined protocol. Services are
loosely coupled to the application and may be distributed.</xs:documentation>
                                </xs:annotation>
                        </xs:enumeration>
                        <xs:enumeration value="Component">
                                <xs:annotation>
                                        <xs:documentation>A nontrivial, nearly independent, and
replaceable part of a system that fulfills a clear function in the context of a well defined architecture. A
component conforms to and provides the physical realizatoin of a set of interfaces. (The Rational Unified
Process Made Easy)</xs:documentation>
                                </xs:annotation>
                        </xs:enumeration>
                        <xs:enumeration value="Library">
                                <xs:annotation>
                                        <xs:documentation>Any collection of pieces of computer
programs that may be used to build other computer programs that do not meet the above
criteria.</xs:documentation>
                                </xs:annotation>
                        </xs:enumeration>
                        <xs:enumeration value="Module/Code Fragment">
                                <xs:annotation>
                                        <xs:documentation>Any piece of a computer program that may be
used to build other computer programs that do not meet the above criteria.</xs:documentation>
                                </xs:annotation>
```

```
				</xs:enumeration>
			</xs:restriction>
		</xs:simpleType>
		<xs:simpleType name="TargetOperatingSystemType">
			<xs:annotation>
				<xs:documentation>Identification of a target operating system applicable to the asset
being submitted. Type: string</xs:documentation>
			</xs:annotation>
			<xs:restriction base="xs:string">
				<xs:minLength value="1"/>
			</xs:restriction>
		</xs:simpleType>
		<xs:simpleType name="TypeAssetType">
			<xs:annotation>
				<xs:documentation>The possible values for identifying the type of asset being
submitted - Documentation or Code</xs:documentation>
			</xs:annotation>
			<xs:restriction base="xs:string">
				<xs:enumeration value="Documentation"/>
				<xs:enumeration value="Code"/>
			</xs:restriction>
		</xs:simpleType>
		<xs:simpleType name="VersionType">
			<xs:annotation>
				<xs:documentation>Version or revision identifier for the asset.</xs:documentation>
			</xs:annotation>
			<xs:restriction base="xs:string">
				<xs:minLength value="1"/>
			</xs:restriction>
		</xs:simpleType>
		<xs:simpleType name="WorkingRelationshipType">
			<xs:annotation>
				<xs:documentation>Possible values for describing the working relationship between
the asset being submitted and other assets - Dependent Upon, Needed By, Interfaces With</xs:documentation>
			</xs:annotation>
			<xs:restriction base="xs:string">
				<xs:enumeration value="Dependent Upon">
					<xs:annotation>
						<xs:documentation>This indicates that the current asset references
or relies on the services or artifacts of the related asset to provide the desired/required
capability.</xs:documentation>
					</xs:annotation>
				</xs:enumeration>
				<xs:enumeration value="Needed By">
					<xs:annotation>
						<xs:documentation>This indicates that the current asset is
referenced or used by the related asset to provide the desired/required capability.</xs:documentation>
					</xs:annotation>
				</xs:enumeration>
				<xs:enumeration value="Interfaces With">
					<xs:annotation>
```

```
                                        <xs:documentation>This indicates that the current asset
communicates with the related assets. The related assets may be needed for operation but that are not needed to
complete the requirements.</xs:documentation>
                                </xs:annotation>
                        </xs:enumeration>
                </xs:restriction>
        </xs:simpleType>
        <xs:simpleType name="YesSelectionType">
                <xs:annotation>
                        <xs:documentation>Enter explanation for the affirmative entry.</xs:documentation>
                </xs:annotation>
                <xs:restriction base="xs:string">
                        <xs:minLength value="1"/>
                </xs:restriction>
        </xs:simpleType>
        <xs:simpleType name="ZipType">
                <xs:annotation>
                        <xs:documentation>String pattern for a 5 or 9 digit zip code. Type:
string</xs:documentation>
                </xs:annotation>
                <xs:restriction base="xs:string">
                        <xs:pattern value="d{5}(-d{4})?"/>
                </xs:restriction>
        </xs:simpleType>
        </xs:schema>
```

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix B. XML Schema for Proposed SHARE Asset Information

This appendix provides two XML schema files, one describing top-level information about Assets (section B.1) and one describing Artifacts (section B.2). The content of these schemas is described in section III.C in the body of this document.

B.1 XML Schema for Describing SHARE Assets

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2008 (http://www.altova.com) by Curtis Blais and Jean Johnson (Naval
Postgraduate School) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.navy.mil/OpenArchitecture/SHARE"
targetNamespace="http://www.navy.mil/OpenArchitecture/SHARE" elementFormDefault="qualified"
attributeFormDefault="unqualified">
	<!-- Include the XML schema describing SHARE Artifacts -->
	<xs:include schemaLocation="SHAREArtifacts.xsd"/>
	<!-- Define the Assets root element -->
	<xs:element name="Assets" type="AssetsType">
		<xs:annotation>
			<xs:documentation>Root element for XML document containing one or more
SHARE Asset records. </xs:documentation>
		</xs:annotation>
	</xs:element>
	<xs:complexType name="AssetsType">
		<xs:annotation>
			<xs:documentation>Complex type defining the data structure for the Assets
element.</xs:documentation>
		</xs:annotation>
		<xs:sequence>
			<xs:element ref="Asset" maxOccurs="unbounded"/>
		</xs:sequence>
	</xs:complexType>
	<xs:element name="Asset" type="AssetType">
		<xs:annotation>
			<xs:documentation>Recommended packaging of artifacts into assets.  Assets are
defined as a grouping of artifacts which provide a solution to a problem for a given context.  Assets can be
either user defined based on search results or packages that are considered common solutions to common
problems.  This differs significantly from the concept of an asset as currently contained in the SHARE
Repository. </xs:documentation>
		</xs:annotation>
	</xs:element>
	<xs:complexType name="AssetType">
		<xs:annotation>
```

```
			<xs:documentation>Complex type defining the data structure for the Asset
element.</xs:documentation>
		</xs:annotation>
		<xs:sequence>
			<xs:element ref="AssetName"/>
			<xs:element ref="Purpose"/>
			<xs:element ref="InitialUse"/>
			<xs:element ref="PreviousUses"/>
			<xs:element ref="AssetScope"/>
			<xs:element ref="AssetCategory"/>
			<xs:element ref="ArtifactsIncluded"/>
			<xs:element ref="RetrievalInformation"/>
		</xs:sequence>
	</xs:complexType>
	<xs:element name="AssetName" type="xs:string">
		<xs:annotation>
			<xs:documentation>Name or identifier of an Asset.</xs:documentation>
		</xs:annotation>
	</xs:element>
	<xs:element name="Purpose" type="xs:string">
		<xs:annotation>
			<xs:documentation>Purpose for the Asset (what the asset does or is used
for).</xs:documentation>
		</xs:annotation>
	</xs:element>
	<xs:element name="InitialUse" type="xs:string">
		<xs:annotation>
			<xs:documentation>Initial use of the Asset.</xs:documentation>
		</xs:annotation>
	</xs:element>
	<xs:element name="PreviousUses" type="PreviousUsesType">
		<xs:annotation>
			<xs:documentation>List of previous uses of the Asset.</xs:documentation>
		</xs:annotation>
	</xs:element>
	<xs:element name="PreviousUse" type="xs:string">
		<xs:annotation>
			<xs:documentation>Description of a previous use of the Asset.</xs:documentation>
		</xs:annotation>
	</xs:element>
	<xs:complexType name="PreviousUsesType">
		<xs:annotation>
			<xs:documentation>Complex type defining the data structure for the PreviousUse
element.</xs:documentation>
		</xs:annotation>
		<xs:sequence>
			<xs:element ref="PreviousUse" maxOccurs="unbounded"/>
		</xs:sequence>
	</xs:complexType>
	<xs:element name="AssetScope" type="AssetScopeType">
		<xs:annotation>
			<xs:documentation>Description of the functional scope of the
Asset.</xs:documentation>
```

```
                </xs:annotation>
        </xs:element>
        <xs:element name="AssetCategory" type="AssetCategoryType">
                <xs:annotation>
                        <xs:documentation>Identification of the category of the Asset.</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:simpleType name="AssetScopeType">
                <xs:annotation>
                        <xs:documentation>Simple type defining the set of enumerated values for
AssetScope.</xs:documentation>
                </xs:annotation>
                <xs:restriction base="xs:string">
                        <xs:enumeration value="Tactical Application"/>
                        <xs:enumeration value="Development Support"/>
                        <xs:enumeration value="Other"/>
                        <xs:enumeration value="Unknown"/>
                </xs:restriction>
        </xs:simpleType>
        <xs:simpleType name="AssetCategoryType">
                <xs:annotation>
                        <xs:documentation>Simple type defining the set of enumerated values for
AssetCategory.</xs:documentation>
                </xs:annotation>
                <xs:restriction base="xs:string">
                        <xs:enumeration value="System"/>
                        <xs:enumeration value="ApplicationProgram"/>
                        <xs:enumeration value="Package"/>
                        <xs:enumeration value="System Service"/>
                        <xs:enumeration value="Component(s)"/>
                        <xs:enumeration value="Library"/>
                        <xs:enumeration value="Module/Code Fragment"/>
                        <xs:enumeration value="Database/Data Files"/>
                        <xs:enumeration value="Framework"/>
                        <xs:enumeration value="Tools/Utilities"/>
                        <xs:enumeration value="Test Tools/Environments"/>
                        <xs:enumeration value="Enterprise Framework"/>
                        <xs:enumeration value="Data Architecture"/>
                        <xs:enumeration value="Pattern/Design/Algorithm"/>
                        <xs:enumeration value="Standard/Interface/API"/>
                </xs:restriction>
        </xs:simpleType>
        <xs:element name="ArtifactsIncluded" type="ArtifactsType">
                <xs:annotation>
                        <xs:documentation>List of Artifacts included in the Asset.</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:element name="RetrievalInformation">
                <xs:annotation>
                        <xs:documentation>Information on how to retrieve the Asset
content.</xs:documentation>
                </xs:annotation>
        </xs:element>
```

```
		</xs:schema>
```

B.2 XML Schema for Describing SHARE Artifacts

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2008 sp1 (http://www.altova.com) by Jean Johnson (Naval Postgraduate School) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.navy.mil/OpenArchitecture/SHARE"
targetNamespace="http://www.navy.mil/OpenArchitecture/SHARE" elementFormDefault="qualified"
attributeFormDefault="unqualified">
		<xs:element name="Artifacts" type="ArtifactsType">
			<xs:annotation>
				<xs:documentation>Artifacts is the root element of the schema - a container for
individual artifacts.  Type: ArtifactsType</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:complexType name="ArtifactsType">
			<xs:annotation>
				<xs:documentation>Contains a list of artifacts.</xs:documentation>
			</xs:annotation>
			<xs:sequence>
				<xs:element ref="Artifact" maxOccurs="unbounded"/>
			</xs:sequence>
		</xs:complexType>
		<xs:element name="Artifact" type="ArtifactType">
			<xs:annotation>
				<xs:documentation>Individual artifacts.  Can either be a full description, or a
reference by physical location or URL.  Type: ArtifactType</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:complexType name="ArtifactType">
			<xs:annotation>
				<xs:documentation>Defines the three possible types of artifact descriptions (full
description, physical reference, or URL reference)</xs:documentation>
			</xs:annotation>
			<xs:choice>
				<xs:element ref="ArtifactFullDescription"/>
				<xs:element ref="ArtifactPhysicalReference"/>
				<xs:element ref="ArtifactURLReference"/>
			</xs:choice>
		</xs:complexType>
		<xs:element name="ArtifactFullDescription" type="ArtifactFullType">
			<xs:annotation>
				<xs:documentation>The recommended full description of an artifact.  Type:
ArtifactFullType</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:element name="ArtifactPhysicalReference" type="ArtifactPhysicalReferenceType">
			<xs:annotation>
				<xs:documentation>A physical reference to an artifact.
Type:ArtifactPhysicalReferenceType</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:element name="ArtifactURLReference" type="URLType">
			<xs:annotation>
				<xs:documentation>A URL reference to an artifact.  Type: 
ArtifactURLReferenceType</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:complexType name="ArtifactFullType">
			<xs:annotation>
				<xs:documentation>Defines the elements of a full artifact 
description.</xs:documentation>
			</xs:annotation>
			<xs:sequence>
				<xs:element ref="ControlNumber"/>
				<xs:element ref="SubmissionInformation"/>
				<xs:element ref="Program"/>
				<xs:element ref="ArtifactDescription"/>
				<xs:element ref="SecurityInformation"/>
				<xs:element ref="DataFormat"/>
				<xs:element ref="RightsRestrictions"/>
				<xs:element ref="AdditionalInformation" minOccurs="0"/>
			</xs:sequence>
		</xs:complexType>
		<xs:simpleType name="ArtifactPhysicalReferenceType">
			<xs:annotation>
				<xs:documentation> Type: string</xs:documentation>
			</xs:annotation>
			<xs:restriction base="xs:string"/>
		</xs:simpleType>
		<xs:simpleType name="URLType">
			<xs:annotation>
				<xs:documentation> Type: string</xs:documentation>
			</xs:annotation>
			<xs:restriction base="xs:string"/>
		</xs:simpleType>
		<xs:element name="ControlNumber" type="ControlNumberType">
			<xs:annotation>
				<xs:documentation>Unique identifacation number for the artifact.  Currently created 
for Assets by the SHARE Domino database.  Type: ControlNumberType </xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:simpleType name="ControlNumberType">
			<xs:annotation>
				<xs:documentation>Type: 32 character string of hexadecimal 
digits</xs:documentation>
			</xs:annotation>
			<xs:restriction base="xs:string">
				<xs:pattern value="[0-9A-F]{32}"/>
			</xs:restriction>
		</xs:simpleType>
		<xs:element name="SubmissionInformation" type="SubmissionInformationType">
			<xs:annotation>
```

```
                <xs:documentation>Artifact submission date, time and source information.  Type: 
SubmissionInformationType</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:complexType name="SubmissionInformationType">
            <xs:annotation>
                <xs:documentation>Defines the elements of the submission 
information.</xs:documentation>
            </xs:annotation>
            <xs:sequence>
                <xs:element ref="SubmitDate"/>
                <xs:element ref="SubmitTime"/>
                <xs:element ref="Source"/>
            </xs:sequence>
        </xs:complexType>
        <xs:element name="SubmitDate" type="xs:date">
            <xs:annotation>
                <xs:documentation>Date of submission.  Type: xs:date</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="SubmitTime" type="xs:time">
            <xs:annotation>
                <xs:documentation>Time of submission:  Type: xs:time</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="Source" type="SourceType">
            <xs:annotation>
                <xs:documentation>Identifies the person and organization responsible for submitting 
the artifact.  Type: SourceType</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:complexType name="SourceType">
            <xs:annotation>
                <xs:documentation>Defines the elements of source 
information.</xs:documentation>
            </xs:annotation>
            <xs:sequence>
                <xs:element ref="ArtifactSource"/>
                <xs:element ref="SourceOrganization"/>
            </xs:sequence>
        </xs:complexType>
        <xs:element name="ArtifactSource" type="POCInfo">
            <xs:annotation>
                <xs:documentation>The individual responsible for submitting the artifact.  
Currently, this information is drawn automatically for the person logged into SHARE conducting the 
submission.  Type: POCInfo</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="SourceOrganization" type="SourceOrganizationType">
            <xs:annotation>
                <xs:documentation>The organization submitting the artifact.  Type: 
SourceOrganizationType </xs:documentation>
            </xs:annotation>
```

```
	</xs:element>
	<xs:complexType name="SourceOrganizationType">
		<xs:annotation>
			<xs:documentation>Defines the elements of information required for the source
organization. The specific information required is dependent on whether or not the organization is a government
or contractor organization.</xs:documentation>
		</xs:annotation>
		<xs:choice>
			<xs:element ref="GovernmentOrganization"/>
			<xs:element ref="ContractorOrganization"/>
		</xs:choice>
	</xs:complexType>
	<xs:element name="GovernmentOrganization" type="GovernmentOrganizationType">
		<xs:annotation>
			<xs:documentation>Identifies a government organization responsible for the
submission.  Type: GovernmentOrganizationType</xs:documentation>
		</xs:annotation>
	</xs:element>
	<xs:complexType name="GovernmentOrganizationType">
		<xs:annotation>
			<xs:documentation>Defines the elements of information required for a government
organization.</xs:documentation>
		</xs:annotation>
		<xs:sequence>
			<xs:element ref="OrganizationName"/>
			<xs:element ref="Patents"/>
		</xs:sequence>
	</xs:complexType>
	<xs:element name="ContractorOrganization" type="ContractorOrganizationType">
		<xs:annotation>
			<xs:documentation>Identifies a contractor organization resposible for the
submission.  Type: ContractorOrganizationType</xs:documentation>
		</xs:annotation>
	</xs:element>
	<xs:complexType name="ContractorOrganizationType">
		<xs:annotation>
			<xs:documentation>Defines the elements of information required for a contractor
organization.</xs:documentation>
		</xs:annotation>
		<xs:sequence>
			<xs:element ref="OrganizationName"/>
			<xs:element ref="ContractNumber"/>
			<xs:element ref="DeliveryVehicle"/>
			<xs:element ref="ContractingOfficer"/>
		</xs:sequence>
	</xs:complexType>
	<xs:element name="OrganizationName" type="xs:string">
		<xs:annotation>
			<xs:documentation>Name of the source organization.  Required for both government
and contractor source organizations.  Type: xs:string</xs:documentation>
		</xs:annotation>
	</xs:element>
	<xs:element name="Patents" type="PatentsType">
```

```
                <xs:annotation>
                        <xs:documentation>A list of relevant patents held by the organization.  Applicable
only to Government source organizations.  Type: PatentsType</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:complexType name="PatentsType">
                <xs:annotation>
                        <xs:documentation>Defines the container for the relevant
patents.</xs:documentation>
                </xs:annotation>
                <xs:sequence>
                        <xs:element name="Patent" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
        <xs:element name="Patent" type="xs:string">
                <xs:annotation>
                        <xs:documentation>Individual patent entries.  Type: xs:stirng</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:element name="ContractNumber" type="xs:string">
                <xs:annotation>
                        <xs:documentation>Contract number the artifact was developed under.  Applicable
only to Contractor source organizations.  Type: xs:string</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:element name="DeliveryVehicle" type="DeliveryVehicleValues">
                <xs:annotation>
                        <xs:documentation>The vehicle of delivery for the artifact.  Applicable only to
Contractor source organizations.  Possible values are DD250, CDRL or Don't Know.  Type:
DeliveryVehicleValues</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:simpleType name="DeliveryVehicleValues">
                <xs:annotation>
                        <xs:documentation>Type: xs:string with possible values of DD250, CDRL, or Don't
Know. </xs:documentation>
                </xs:annotation>
                <xs:restriction base="xs:string">
                        <xs:enumeration value="DD250"/>
                        <xs:enumeration value="CDRL"/>
                        <xs:enumeration value="Don't Know"/>
                </xs:restriction>
        </xs:simpleType>
        <xs:element name="ContractingOfficer" type="POCInfo">
                <xs:annotation>
                        <xs:documentation>POC Information for the contracting officer on the contract.
Applicable only to Contractor source organizations.  Type: POCInfo</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:element name="Program" type="ProgramType">
                <xs:annotation>
                        <xs:documentation>Name and manager information for the major program
responsible for development of the artifact.  Type: ProgramType</xs:documentation>
```

```
                </xs:annotation>
        </xs:element>
        <xs:complexType name="ProgramType">
                <xs:annotation>
                        <xs:documentation>Defines the elements of information required for the
program.</xs:documentation>
                </xs:annotation>
                <xs:sequence>
                        <xs:element ref="ProgramName"/>
                        <xs:element ref="ProgramManager"/>
                </xs:sequence>
        </xs:complexType>
        <xs:element name="ProgramName" type="ProgramNameType">
                <xs:annotation>
                        <xs:documentation>Name of program.  Can either be the name of a program existing
in SHARE, or one to be added.  Type: ProgramNameType</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:complexType name="ProgramNameType">
                <xs:annotation>
                        <xs:documentation>Defines two types of program names, existing or
new.</xs:documentation>
                </xs:annotation>
                <xs:choice>
                        <xs:element ref="ExistingProgramName"/>
                        <xs:element ref="NewProgramName"/>
                </xs:choice>
        </xs:complexType>
        <xs:element name="ExistingProgramName" type="ProgramNameValues">
                <xs:annotation>
                        <xs:documentation>Existing program names as enumerated by a list.  Type:
ProgramNameValues</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:simpleType name="ProgramNameValues">
                <xs:annotation>
                        <xs:documentation>Type: xs:string, with an enumberated list of possible values for
ExistingProgramName.</xs:documentation>
                </xs:annotation>
                <xs:restriction base="xs:string">
                        <xs:enumeration value="AEGIS"/>
                        <xs:enumeration value="DDG 1000"/>
                        <xs:enumeration value="SSDS"/>
                        <xs:enumeration value="LCS"/>
                        <xs:enumeration value="NSWCDD HSI"/>
                        <xs:enumeration value="SIAP"/>
                        <xs:enumeration value="SQQ-89"/>
                        <xs:enumeration value="TSTS"/>
                        <xs:enumeration value="ASW"/>
                        <xs:enumeration value="CEP WASP"/>
                        <xs:enumeration value="GeDear"/>
                        <xs:enumeration value="BFTT"/>
                </xs:restriction>
        </xs:simpleType>
        <xs:element name="NewProgramName" type="xs:string">
                <xs:annotation>
                        <xs:documentation>Name of program when it is not already included in existing
SHARE programs.  Type: xs:string</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:element name="ProgramManager" type="POCInfo">
                <xs:annotation>
                        <xs:documentation>POC information for the program manager.  When the program
already exists in SHARE, this information is automatically drawn from the database.  Otherwise, a submitting
user is prompted to enter the information.  Type: POCInfo</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:element name="ArtifactDescription" type="ArtifactDescriptionType">
                <xs:annotation>
                        <xs:documentation>The portion of the full artifact description that focuses on the
artifact itself.  Type: ArtifactDescriptionType</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:complexType name="ArtifactDescriptionType">
                <xs:annotation>
                        <xs:documentation>Defines the two types of artifact descriptions: NonCode and
Code.</xs:documentation>
                </xs:annotation>
                <xs:choice>
                        <xs:element ref="NonCodeDescription"/>
                        <xs:element ref="CodeDescription"/>
                </xs:choice>
        </xs:complexType>
        <xs:element name="NonCodeDescription" type="NonCodeDescriptionType">
                <xs:annotation>
                        <xs:documentation>The information required for any type of artifact other than
code.  Type: NonCodeDescriptionType</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:complexType name="NonCodeDescriptionType">
                <xs:annotation>
                        <xs:documentation>Defines the information required for non-code artfacts. Consists
of a group of elements defined by the ItemDescriptionGroup.</xs:documentation>
                </xs:annotation>
                <xs:group ref="ItemDescriptionGroup"/>
        </xs:complexType>
        <xs:group name="ItemDescriptionGroup">
                <xs:annotation>
                        <xs:documentation>Defines a group of elements that describe both code and non-
code artifacts.  </xs:documentation>
                </xs:annotation>
                <xs:sequence>
                        <xs:element ref="ArtifactName"/>
                        <xs:element ref="Version"/>
                        <xs:element ref="DateOfCreation"/>
                        <xs:element ref="Description"/>
```

```
                    <xs:element ref="ContributionRationale" minOccurs="0"/>
                    <xs:element ref="ArtifactType"/>
                    <xs:element ref="ApplicableSystems"/>
                    <xs:element ref="ObjectiveArchitectureTags"/>
                    <xs:element ref="SoftwareBehaviorDescription"/>
                    <xs:element ref="History"/>
                    <xs:element ref="Interdependencies"/>
               </xs:sequence>
          </xs:group>
          <xs:element name="ArtifactName" type="xs:string">
               <xs:annotation>
                    <xs:documentation>Artifact name. Type: xs:string</xs:documentation>
               </xs:annotation>
          </xs:element>
          <xs:element name="Version" type="xs:string">
               <xs:annotation>
                    <xs:documentation>Artifact version.  Type: xs:string</xs:documentation>
               </xs:annotation>
          </xs:element>
          <xs:element name="DateOfCreation" type="xs:date">
               <xs:annotation>
                    <xs:documentation>Date of artifact creation (may be approximated).  Type:
xs:date</xs:documentation>
               </xs:annotation>
          </xs:element>
          <xs:element name="Description" type="xs:string">
               <xs:annotation>
                    <xs:documentation>A brief description of the artifact provided by the submitter.
Type: xs:sring</xs:documentation>
               </xs:annotation>
          </xs:element>
          <xs:element name="ContributionRationale" type="xs:string">
               <xs:annotation>
                    <xs:documentation>The submitter's rationale for submitting the artifact.  Type:
xs:string</xs:documentation>
               </xs:annotation>
          </xs:element>
          <xs:element name="ArtifactType" type="ArtifactTypeValues">
               <xs:annotation>
                    <xs:documentation>Artifact type and subtypes are identified as attributes of this
element.  Type: ArtifactTypeValues</xs:documentation>
               </xs:annotation>
          </xs:element>
          <xs:complexType name="ArtifactTypeValues">
               <xs:annotation>
                    <xs:appinfo source="artifacts.owl">
                         The allowable values for Type and SubType should be validated against the
SoftwareArtifacts classes and their subclasses (respectively) from the artifact-lifecycle ontology developed for
this project.
                    </xs:appinfo>
                    <xs:documentation>The possible values for artifact type.  These should be validated
against the artifact-lifecycle ontology developed for this project found in separate document
"artifacts.owl"</xs:documentation>
```

```
				</xs:annotation>
				<xs:attribute ref="Type"/>
				<xs:attribute ref="SubType"/>
			</xs:complexType>
			<xs:attribute name="Type" type="xs:string">
				<xs:annotation>
					<xs:documentation>Corresponds to the subclasses of the SoftwareArtifacts class in 
the artifacts-lifecycle ontology.</xs:documentation>
				</xs:annotation>
			</xs:attribute>
			<xs:attribute name="SubType" type="xs:string">
				<xs:annotation>
					<xs:documentation>Corresponds to the second level of subclasses under the 
SoftwareArtifacts class in the artifacts-lifecycle ontology.</xs:documentation>
				</xs:annotation>
			</xs:attribute>
			<xs:element name="ApplicableSystems" type="ApplicableSystemsType">
				<xs:annotation>
					<xs:documentation>The systems in which the artirfact is currently used.  Type: 
ApplicableSystemsType</xs:documentation>
				</xs:annotation>
			</xs:element>
			<xs:complexType name="ApplicableSystemsType">
				<xs:annotation>
					<xs:documentation>A list of systems/subsystems in which the artifact is used.  
Ideally, these system/subsystem groupings could be validated against ontologies that are based on the system 
diagrams for systems represented in SHARE.  These diagrams were not available for ontology development for 
this version of the repository framework.</xs:documentation>
				</xs:annotation>
				<xs:sequence maxOccurs="unbounded">
					<xs:element ref="System"/>
					<xs:element ref="Subsystem" minOccurs="0"/>
				</xs:sequence>
			</xs:complexType>
			<xs:element name="System" type="xs:string">
				<xs:annotation>
					<xs:documentation>The major system in which the artifact is used.  Type: 
xs:string</xs:documentation>
				</xs:annotation>
			</xs:element>
			<xs:element name="Subsystem" type="xs:string">
				<xs:annotation>
					<xs:documentation>The smallest subsystem in which the artifact is used.  This is an 
optional field since an artifact may apply to a complete system. Type: xs:string</xs:documentation>
				</xs:annotation>
			</xs:element>
			<xs:element name="ObjectiveArchitectureTags" type="ObjectiveArchitectureTagsType">
				<xs:annotation>
					<xs:documentation>The portion of the Navy's objective architecture to which the 
artifact applies.  Type: ObjectiveArchitectureTagsType</xs:documentation>
				</xs:annotation>
			</xs:element>
			<xs:complexType name="ObjectiveArchitectureTagsType">
			<xs:annotation>
				<xs:documentation>A list of the domain/subdomain elements of the objective
architecture to which the artifact applies.  These should be validated against the ontology built for this project
based on the Surface Combat System Objective Architecture developed for the Naval Open Architecture
project.  This ontology is found in separate document
"SurfaceWarfareSystemObjectiveArchitectureTagsType.owl".</xs:documentation>
			</xs:annotation>
			<xs:sequence>
				<xs:annotation>
					<xs:appinfo source="SurfaceWarfareSystemObjectiveArchitecture.owl">
						The allowable values for DomainTags and SubsystemTags should
be validated against the PlatformAdaptation subclasses (domains) and their subclasses from the ontology
created for this project from the Surface Combat System Objective Architecture developed for the Naval Open
Architecture project.
					</xs:appinfo>
				</xs:annotation>
				<xs:element ref="DomainTags"/>
				<xs:element ref="SubDomainTags"/>
			</xs:sequence>
		</xs:complexType>
		<xs:element name="DomainTags" type="DomainTagsType">
			<xs:annotation>
				<xs:documentation>The domain elements in the objective architecture to which the
artifact applies.  Should be verified against the subclasses of the PlatformAdaptation class in the objective
architecture ontology.  Type: DomainTagsType</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:complexType name="DomainTagsType">
			<xs:annotation>
				<xs:documentation>A list of domain elements relevant for the
artifact.</xs:documentation>
			</xs:annotation>
			<xs:sequence>
				<xs:element ref="DomainTag" maxOccurs="unbounded"/>
			</xs:sequence>
		</xs:complexType>
		<xs:element name="DomainTag" type="xs:string">
			<xs:annotation>
				<xs:documentation>Individual domain elements.  Type:
xs:string</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:element name="SubDomainTags" type="SubDomainTagsType">
			<xs:annotation>
				<xs:documentation>The subdomain elements in the objective architecture to which
the artifact applies.  Should be verified against the second level subclasses of the PlatformAdaptation class in
the PlatformAdaptation class of the objective architecture ontology.  Type:
SubDomainTagsType</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:complexType name="SubDomainTagsType">
			<xs:annotation>
```

```
            <xs:documentation>A list of subdomain elements relevant for the
artifact.</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element ref="SubDomainTag" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="SubDomainTag" type="xs:string">
        <xs:annotation>
            <xs:documentation>Individual subdomain elements.  Type:
xs:string</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="SoftwareBehaviorDescription" type="SoftwareBehaviorDescriptionType">
        <xs:annotation>
            <xs:documentation>A description of the behavior of the software found in or related
to the artifact.  Type: SoftwareBehaviorDescriptionType</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="SoftwareBehaviorDescriptionType">
        <xs:annotation>
            <xs:documentation>Defines the two possible parts of the software behavior
description, the CSFL items and the WSDL description.</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element ref="CommonSystemFunctionList"/>
            <xs:element ref="WebServicesDescription" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="CommonSystemFunctionList" type="CSFLType">
        <xs:annotation>
            <xs:documentation>Functions that are addressed by the artifact, validated against the
ontology built for this project based on the Navy's CSFL found in separate document,
"CommonSystemFunctionList.owl". Type: CSFLType</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="CSFLType">
        <xs:annotation>
            <xs:documentation>A list of functions that are addressed by the
artifact.</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:annotation>
                <!--Placeholder for now until CSFL ontology is complete -->
                <xs:appinfo source="CommonSystemFunctionList.owl">
                    The allowable values for each Common System Function should
be validated against the CSFL ontology created for this project.
                </xs:appinfo>
            </xs:annotation>
            <xs:element ref="CommonSystemFunction" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
```

```
        <xs:element name="CommonSystemFunction" type="xs:string">
                <xs:annotation>
                        <xs:documentation>Individual functions.  Currently expressed as an optional item
until the information can be input for all artifacts in SHARE.  Type: xs:string</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:element name="WebServicesDescription" type="xs:string">
                <xs:annotation>
                        <xs:documentation>The Web Services Description Language (WSDL) description
of the software found in or related to the artifact.  Currently a placeholder until better defined  Type:
xs:string</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:element name="History" type="HistoryType">
                <xs:annotation>
                        <xs:documentation>The history/pedigree of the artifact.  Type:
HistoryType</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:complexType name="HistoryType">
                <xs:annotation>
                        <xs:documentation>Defines the elements that make up the history of
artifact.</xs:documentation>
                </xs:annotation>
                <xs:sequence>
                        <xs:element ref="DevelopmentStatus"/>
                        <xs:element ref="PlannedUpdates" minOccurs="0"/>
                        <xs:element ref="MaturityDescription" minOccurs="0"/>
                        <xs:element ref="Pedigree"/>
                </xs:sequence>
        </xs:complexType>
        <xs:element name="DevelopmentStatus" type="DevelopmentStatusValues">
                <xs:annotation>
                        <xs:documentation>Indication of whether the artifact is in development or complete.
Type: DevelopmentStatusValues</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:simpleType name="DevelopmentStatusValues">
                <xs:annotation>
                        <xs:documentation>The possible values for development status - InDevelopment or
DevelopmentComplete.</xs:documentation>
                </xs:annotation>
                <xs:restriction base="xs:string">
                        <xs:enumeration value="InDevelopment"/>
                        <xs:enumeration value="DevelopmentComplete"/>
                </xs:restriction>
        </xs:simpleType>
        <xs:element name="PlannedUpdates" type="xs:string">
                <xs:annotation>
                        <xs:documentation>Indication of any known planned updates for the artifact.  Type:
xs:string</xs:documentation>
                </xs:annotation>
        </xs:element>
```

```
		<xs:element name="MaturityDescription" type="xs:string">
			<xs:annotation>
				<xs:documentation>A submitter-defined description of the maturity of the artifact. 
Type: xs:string</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:element name="Pedigree" type="PedigreeType">
			<xs:annotation>
				<xs:documentation>Part of an artifact's history, the pedigree shows how artifacts 
may have evolved from each other and how they may be dependent on each other.  </xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:element name="NewerVersionOf" type="xs:string">
			<xs:annotation>
				<xs:documentation>Lists any artifacts for which the current artifact is a newer 
version.  Current artifact is intended to upgrade or replace these listed artifacts.</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:element name="VariantOf" type="xs:string">
			<xs:annotation>
				<xs:documentation>Lists any artifacts for which the current artifact is a variant to be 
used in a different target environment or situation.</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:element name="ExtractedFrom" type="xs:string">
			<xs:annotation>
				<xs:documentation>Lists any artifacts that the current artifact was extracted from 
and possibly packaged to be more reusable.  </xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:element name="DerivedFrom" type="xs:string">
			<xs:annotation>
				<xs:documentation>LIsts any artifacts from which the current artifact was derived. 
</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:element name="OlderVersion" type="xs:string">
			<xs:annotation>
				<xs:documentation>Lists any artifacts that are updates or replacements for the 
current artifact.</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:element name="ExtractionSource" type="xs:string">
			<xs:annotation>
				<xs:documentation>Lists any artifacts for which the current artifact was used to 
extract the related artifact.</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:element name="DerivationSource" type="xs:string">
			<xs:annotation>
				<xs:documentation>Lists any artifacts for which the current artifact was used to 
derive the related artifact.</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:element name="SimilarTo" type="xs:string">
			<xs:annotation>
				<xs:documentation>Lists any artifacts with similar characteristics to the current
artifact.</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:element name="ContainedWithin" type="xs:string">
			<xs:annotation>
				<xs:documentation>Lists any artifacts that contain the current artifact physically or
by reference.</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:element name="Contains" type="xs:string">
			<xs:annotation>
				<xs:documentation>Lists any artifacts that the current artifact contains physically or
by reference.</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:complexType name="PedigreeType">
			<xs:annotation>
				<xs:documentation>Defines the elements contained in an artifact's
pedigree.</xs:documentation>
			</xs:annotation>
			<xs:sequence>
				<xs:element ref="NewerVersionOf" minOccurs="0" maxOccurs="unbounded"/>
				<xs:element ref="VariantOf" minOccurs="0" maxOccurs="unbounded"/>
				<xs:element ref="ExtractedFrom" minOccurs="0" maxOccurs="unbounded"/>
				<xs:element ref="DerivedFrom" minOccurs="0" maxOccurs="unbounded"/>
				<xs:element ref="OlderVersion" minOccurs="0" maxOccurs="unbounded"/>
				<xs:element ref="ExtractionSource" minOccurs="0" maxOccurs="unbounded"/>
				<xs:element ref="DerivationSource" minOccurs="0" maxOccurs="unbounded"/>
				<xs:element ref="SimilarTo" minOccurs="0" maxOccurs="unbounded"/>
				<xs:element ref="ContainedWithin" minOccurs="0" maxOccurs="unbounded"/>
				<xs:element ref="Contains" minOccurs="0" maxOccurs="unbounded"/>
			</xs:sequence>
		</xs:complexType>
		<xs:element name="Interdependencies" type="InterdependenciesType">
			<xs:annotation>
				<xs:documentation>Defines possible interdependencies of
artifacts.</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:complexType name="InterdependenciesType">
			<xs:annotation>
				<xs:documentation>Defines the elements contained in the artifact's
interdependencies.</xs:documentation>
			</xs:annotation>
			<xs:sequence>
				<xs:element ref="DependentUpon" minOccurs="0" maxOccurs="unbounded"/>
				<xs:element ref="NeededBy" minOccurs="0" maxOccurs="unbounded"/>
				<xs:element ref="COTS_GOTSDependencies" minOccurs="0"
maxOccurs="unbounded"/>
```

```
				<xs:element ref="InterfacesWith" minOccurs="0" maxOccurs="unbounded"/>
			</xs:sequence>
		</xs:complexType>
		<xs:element name="DependentUpon" type="xs:string">
			<xs:annotation>
				<xs:documentation>Lists artifacts that the current arftifact relies on to provide the 
desired/required capability.</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:element name="NeededBy" type="xs:string">
			<xs:annotation>
				<xs:documentation>Lists artifacts that rely on the current artifact to provide 
desired/required capability.</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:element name="COTS_GOTSDependencies" type="xs:string">
			<xs:annotation>
				<xs:documentation>Lists any COTS_GOTS dependencies of the current 
artifact.</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:element name="InterfacesWith" type="xs:string">
			<xs:annotation>
				<xs:documentation>Lists artifacts with which the current artifact 
communicates.</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:element name="CodeDescription" type="CodeDescriptionType">
			<xs:annotation>
				<xs:documentation>The information required for code artifacts.  Type: 
CodeDescriptionType</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:complexType name="CodeDescriptionType">
			<xs:annotation>
				<xs:documentation>Defines the information required for code artifacts.  Includes the 
ItemDescriptionGroup elements defined for non-code artifacts plus several additional elements.
</xs:documentation>
			</xs:annotation>
			<xs:sequence>
				<xs:group ref="ItemDescriptionGroup"/>
				<xs:element ref="TargetOS"/>
				<xs:element ref="ProgrammingLanguages"/>
				<xs:element ref="RuntimeEnvironments"/>
				<xs:element ref="KSLOC"/>
				<xs:element ref="TotalLinesOfComment"/>
			</xs:sequence>
		</xs:complexType>
		<xs:element name="TargetOS" type="xs:string">
			<xs:annotation>
				<xs:documentation>Target Operating System for the artifact  Type: 
xs:string</xs:documentation>
			</xs:annotation>
```

```
        </xs:element>
        <xs:element name="ProgrammingLanguages" type="xs:string">
                <xs:annotation>
                        <xs:documentation>Programming Languages used for the artifact.  Type: 
xs:string</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:element name="RuntimeEnvironments" type="xs:string">
                <xs:annotation>
                        <xs:documentation>Runtime environments intended for the artifact.  Type: 
xs:string</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:element name="KSLOC" type="xs:positiveInteger">
                <xs:annotation>
                        <xs:documentation>Thousand source lines of code (KSLOC) in the artifact.  Type: 
xs:positiveInteger</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:element name="TotalLinesOfComment" type="xs:positiveInteger">
                <xs:annotation>
                        <xs:documentation>Total lines of comment in the artifact.  Type: 
xs:positiveInteger</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:element name="SecurityInformation" type="SecurityInformationType">
                <xs:annotation>
                        <xs:documentation>The classification information required for the artifact.  Type: 
SecurityInformationType</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:complexType name="SecurityInformationType">
                <xs:annotation>
                        <xs:documentation>Defines the elements included in the security information of the 
artifact.</xs:documentation>
                </xs:annotation>
                <xs:choice>
                        <xs:element ref="Unclassified"/>
                        <xs:element ref="Classified"/>
                </xs:choice>
        </xs:complexType>
        <xs:element name="Unclassified" type="UnclassifiedInformationType">
                <xs:annotation>
                        <xs:documentation>The information required for unclassified artifacts.  Type: 
UnclassifiedInformationType</xs:documentation>
                </xs:annotation>
        </xs:element>
        <xs:complexType name="UnclassifiedInformationType">
                <xs:annotation>
                        <xs:documentation>Defines the information required for unclassified artifacts. 
Includes the elements in the ClassificationInformationGroup</xs:documentation>
                </xs:annotation>
                <xs:group ref="ClassificationInformationGroup"/>
```

```
		</xs:complexType>
		<xs:group name="ClassificationInformationGroup">
			<xs:annotation>
				<xs:documentation>The group of classsification elements required for artirfacts of
all security levels.</xs:documentation>
			</xs:annotation>
			<xs:sequence>
				<xs:element ref="Classification"/>
				<xs:element ref="ExportControl"/>
				<xs:element ref="DistributionStatement"/>
			</xs:sequence>
		</xs:group>
		<xs:element name="Classified" type="ClassifiedInformationType">
			<xs:annotation>
				<xs:documentation>The information required for classified artifacts.  Type:
ClassifiedInformationType</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:complexType name="ClassifiedInformationType">
			<xs:annotation>
				<xs:documentation>Defines the information required for classified artifacts.
Includes the elements in the ClassificationInformationGroup plus the Classification Guide
ID.</xs:documentation>
			</xs:annotation>
			<xs:sequence>
				<xs:group ref="ClassificationInformationGroup"/>
				<xs:element ref="ClassificationGuideID"/>
			</xs:sequence>
		</xs:complexType>
		<xs:element name="Classification" type="ClassificationValues">
			<xs:annotation>
				<xs:documentation>The classification of the artifact.  Type:
ClassificationValues</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:simpleType name="ClassificationValues">
			<xs:annotation>
				<xs:documentation>The possible values for the classification of the artifact - U, C,
or S</xs:documentation>
			</xs:annotation>
			<xs:restriction base="xs:string">
				<xs:enumeration value="U"/>
				<xs:enumeration value="C"/>
				<xs:enumeration value="S"/>
			</xs:restriction>
		</xs:simpleType>
		<xs:element name="ExportControl" type="YesNo">
			<xs:annotation>
				<xs:documentation>Indication of whether or not the artifact is subject to export
control.  Type: YesNo</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:element name="DistributionStatement" type="DistributionStatementValues">
```

```
				<xs:annotation>
					<xs:documentation>The distribution statement applicable for the artifact.  Type: 
DistributionStatementValues</xs:documentation>
				</xs:annotation>
			</xs:element>
			<xs:simpleType name="DistributionStatementValues">
				<xs:annotation>
					<xs:documentation>The possible values for the distribution statement applicable to 
the artifact - A, B, C, D, E, F, or X</xs:documentation>
				</xs:annotation>
				<xs:restriction base="xs:string">
					<xs:enumeration value="A"/>
					<xs:enumeration value="B"/>
					<xs:enumeration value="C"/>
					<xs:enumeration value="D"/>
					<xs:enumeration value="E"/>
					<xs:enumeration value="F"/>
					<xs:enumeration value="X"/>
				</xs:restriction>
			</xs:simpleType>
			<xs:element name="ClassificationGuideID" type="xs:string">
				<xs:annotation>
					<xs:documentation>The applicable Security Classification Guide for the artifact.  
Applicable only to classified artifacts.  Type: xs:string</xs:documentation>
				</xs:annotation>
			</xs:element>
			<xs:element name="DataFormat" type="DataFormatType">
				<xs:annotation>
					<xs:documentation>The information related to the artifact's format.  Type: 
DataFormatType</xs:documentation>
				</xs:annotation>
			</xs:element>
			<xs:complexType name="DataFormatType">
				<xs:annotation>
					<xs:documentation>Defines the elements required related to the artifact's 
format.</xs:documentation>
				</xs:annotation>
				<xs:sequence>
					<xs:element ref="PhysicalMediaFormat" minOccurs="0"/>
					<xs:element ref="FileNames"/>
					<xs:element ref="ArchiveFormats" minOccurs="0"/>
					<xs:element ref="TotalDataSize" minOccurs="0"/>
				</xs:sequence>
			</xs:complexType>
			<xs:element name="PhysicalMediaFormat" type="PhysicalMediaFormatValues">
				<xs:annotation>
					<xs:documentation>The physical media of the format (optional).  Type: 
PhysicalMediaFormatValues</xs:documentation>
				</xs:annotation>
			</xs:element>
			<xs:simpleType name="PhysicalMediaFormatValues">
				<xs:annotation>
```

```
					<xs:documentation>The possible values for the media format - CD or 
DVD</xs:documentation>
				</xs:annotation>
				<xs:restriction base="xs:string">
					<xs:enumeration value="CD"/>
					<xs:enumeration value="DVD"/>
				</xs:restriction>
			</xs:simpleType>
			<xs:element name="FileNames" type="FileNamesType">
				<xs:annotation>
					<xs:documentation>The names of the files included in the artifact.  Type 
FIleNamesType</xs:documentation>
				</xs:annotation>
			</xs:element>
			<xs:complexType name="FileNamesType">
				<xs:annotation>
					<xs:documentation>A list of the names of the files included in the 
artifact.</xs:documentation>
				</xs:annotation>
				<xs:sequence>
					<xs:element ref="FileName" maxOccurs="unbounded"/>
				</xs:sequence>
			</xs:complexType>
			<xs:element name="FileName" type="xs:string">
				<xs:annotation>
					<xs:documentation>Individual file names.  Type: xs:string</xs:documentation>
				</xs:annotation>
			</xs:element>
			<xs:element name="ArchiveFormats" type="xs:string">
				<xs:annotation>
					<xs:documentation>Archive formats (optional).  Type: 
xs:string</xs:documentation>
				</xs:annotation>
			</xs:element>
			<xs:element name="TotalDataSize" type="xs:string">
				<xs:annotation>
					<xs:documentation>Total size of the artifact files (optional).  Type: 
xs:string</xs:documentation>
				</xs:annotation>
			</xs:element>
			<xs:element name="RightsRestrictions" type="RightsRestrictionsType">
				<xs:annotation>
					<xs:documentation>The information related to any rights restrictions on the artifact. 
Type: RightsRestrictionsType</xs:documentation>
				</xs:annotation>
			</xs:element>
			<xs:complexType name="RightsRestrictionsType">
				<xs:annotation>
					<xs:documentation>The elements considered under rights restrictions for the artifact.
</xs:documentation>
				</xs:annotation>
				<xs:sequence>
					<xs:element ref="DataRightsMarkings"/>
					<xs:element ref="CommercialSoftware"/>
					<xs:element ref="SpecialLicenses"/>
					<xs:element ref="OpenSourceSoftwareLicenses"/>
					<xs:element ref="DataRightsAssertions"/>
				</xs:sequence>
			</xs:complexType>
			<xs:element name="DataRightsMarkings" type="IncludedType">
				<xs:annotation>
					<xs:documentation>Data rights markings on the artifact.  Includes a YesNo
"Included" attribute as well as free text describing the markings.  Type: IncludedType</xs:documentation>
				</xs:annotation>
			</xs:element>
			<xs:element name="CommercialSoftware" type="IncludedType">
				<xs:annotation>
					<xs:documentation>Commercial software used in the artifact.  Includes a YesNo
"Included" attribute as well as free text describing the commercial software.  Type:
IncludedType</xs:documentation>
				</xs:annotation>
			</xs:element>
			<xs:element name="SpecialLicenses" type="IncludedType">
				<xs:annotation>
					<xs:documentation>Special licenses placed on the artifact.  Includes a YesNo
"Included" attribute as well as free text describing the licenses.  Type: IncludedType</xs:documentation>
				</xs:annotation>
			</xs:element>
			<xs:element name="OpenSourceSoftwareLicenses" type="IncludedType">
				<xs:annotation>
					<xs:documentation>Open source software licenses placed on the artifact.  Includes a
YesNo "Included" attribute as well as free text describing the licenses.  Type:
IncludedType</xs:documentation>
				</xs:annotation>
			</xs:element>
			<xs:element name="DataRightsAssertions" type="IncludedType">
				<xs:annotation>
					<xs:documentation>Data rights assertions on the artifact.  Includes a YesNo
"Included" attribute as well as free text describing the assertions.  Type: IncludedType</xs:documentation>
				</xs:annotation>
			</xs:element>
			<xs:complexType name="IncludedType">
				<xs:annotation>
					<xs:documentation>Type allowing a yes/no indication for inclusion as well as free
text explanation.</xs:documentation>
				</xs:annotation>
				<xs:simpleContent>
					<xs:extension base="xs:string">
						<xs:attribute ref="Included"/>
					</xs:extension>
				</xs:simpleContent>
			</xs:complexType>
			<xs:attribute name="Included" type="YesNo">
				<xs:annotation>
					<xs:documentation>Attribute indicating whether or not the restriction is applicable
for the artifact.  Type:  YesNo</xs:documentation>
```

```
            </xs:annotation>
        </xs:attribute>
        <xs:element name="AdditionalInformation" type="xs:string">
            <xs:annotation>
                <xs:documentation>Text allowance for any additional information about the artifact
deemed important. Type: xs:string</xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:complexType name="POCInfo">
            <xs:annotation>
                <xs:documentation>Required information for points of contact.</xs:documentation>
            </xs:annotation>
            <xs:sequence>
                <xs:element name="Name" type="xs:string"/>
                <xs:element name="Organization" type="xs:string"/>
                <xs:element name="MailingAddress" type="AddressType"/>
                <xs:element name="Phone" type="PhoneNumber"/>
                <xs:element name="Email" type="EmailAddress"/>
            </xs:sequence>
        </xs:complexType>
        <xs:simpleType name="PhoneNumber">
            <xs:annotation>
                <xs:documentation>Phone number</xs:documentation>
            </xs:annotation>
            <xs:restriction base="xs:string">
                <xs:pattern value="d{3} d{3}-d{4} x?(d{4})?"/>
            </xs:restriction>
        </xs:simpleType>
        <xs:simpleType name="EmailAddress">
            <xs:annotation>
                <xs:documentation>Email address</xs:documentation>
            </xs:annotation>
            <xs:restriction base="xs:string">
                <xs:pattern value=".+@.+"/>
            </xs:restriction>
        </xs:simpleType>
        <xs:complexType name="AddressType">
            <xs:annotation>
                <xs:documentation>Address </xs:documentation>
            </xs:annotation>
            <xs:sequence>
                <xs:element name="AddressLine1" type="xs:string"/>
                <xs:element name="AddressLine2" minOccurs="0"/>
                <xs:element name="City" type="xs:string"/>
                <xs:element name="State" type="xs:string"/>
                <xs:element name="Zip" type="ZipCode"/>
            </xs:sequence>
        </xs:complexType>
        <xs:simpleType name="ZipCode">
            <xs:annotation>
                <xs:documentation>Zip Code</xs:documentation>
            </xs:annotation>
            <xs:restriction base="xs:string">
```

```
				<xs:pattern value="d{5}-?(d{4})?"/>
			</xs:restriction>
		</xs:simpleType>
		<xs:simpleType name="YesNo">
			<xs:annotation>
				<xs:documentation>Possible values for YesNo elements type.</xs:documentation>
			</xs:annotation>
			<xs:restriction base="xs:string">
				<xs:enumeration value="Yes"/>
				<xs:enumeration value="No"/>
			</xs:restriction>
		</xs:simpleType>
</xs:schema>
```

THIS PAGE INTENTIONALLY LEFT BLANK

2003 - 2008 Sponsored Research Topics

Acquisition Management

- Software Requirements for OA
- Managing Services Supply Chain
- Acquiring Combat Capability via Public-Private Partnerships (PPPs)
- Knowledge Value Added (KVA) + Real Options (RO) Applied to Shipyard Planning Processes
- Portfolio Optimization via KVA + RO
- MOSA Contracting Implications
- Strategy for Defense Acquisition Research
- Spiral Development
- BCA: Contractor vs. Organic Growth

Contract Management

- USAF IT Commodity Council
- Contractors in 21st Century Combat Zone
- Joint Contingency Contracting
- Navy Contract Writing Guide
- Commodity Sourcing Strategies
- Past Performance in Source Selection
- USMC Contingency Contracting
- Transforming DoD Contract Closeout
- Model for Optimizing Contingency Contracting Planning and Execution

Financial Management

- PPPs and Government Financing
- Energy Saving Contracts/DoD Mobile Assets
- Capital Budgeting for DoD
- Financing DoD Budget via PPPs
- ROI of Information Warfare Systems
- Acquisitions via leasing: MPS case
- Special Termination Liability in MDAPs

Human Resources

- Learning Management Systems
- Tuition Assistance
- Retention
- Indefinite Reenlistment
- Individual Augmentation

Logistics Management

- R-TOC Aegis Microwave Power Tubes
- Privatization-NOSL/NAWCI
- Army LOG MOD
- PBL (4)
- Contractors Supporting Military Operations
- RFID (4)
- Strategic Sourcing
- ASDS Product Support Analysis
- Analysis of LAV Depot Maintenance
- Diffusion/Variability on Vendor Performance Evaluation
- Optimizing CIWS Lifecycle Support (LCS)

Program Management

- Building Collaborative Capacity
- Knowledge, Responsibilities and Decision Rights in MDAPs
- KVA Applied to Aegis and SSDS
- Business Process Reengineering (BPR) for LCS Mission Module Acquisition
- Terminating Your Own Program
- Collaborative IT Tools Leveraging Competence

A complete listing and electronic copies of published research are available on our website: www.acquisitionresearch.org