Technical Report: SysML v1 to SysML v2 Model Conversion Approach

March 2024

Version 1.2

Office of Systems Engineering and Architecture

Office of the Under Secretary of Defense for Research and Engineering

Washington, D.C.

Abstract

Systems Modeling Language (SysML) Version 2 (v2) is the next generation of SysML, a standard modeling language used in systems engineering throughout industry and government. The Object Management Group (OMG), an international consortium, adopted SysML Version 1 (v1) in 2007 and is expected to adopt SysML v2 in 2024.

SysML v2 supports the evolving practice of model-based systems engineering (MBSE) to address challenges in increasing system complexity and rapid technology change. SysML v2 is intended to increase the effectiveness and adoption of MBSE by improving the language precision, expressiveness, regularity, and interoperability.

This paper presents a suggested approach to make the transition from SysML v1 to SysML v2. The approach was developed as part of a project sponsored by the Department of Defense (DoD) Office of the Under Secretary of Defense for Research and Engineering, Systems Engineering and Architecture (SE&A), Digital Engineering, Modeling and Simulation (DEM&S). To illustrate the transition, the project team used a hypothetical system model, Skyzer Mission Model, developed by the Systems Engineering Research Center, a DoD university-affiliated research center.

SE&A may revise this information as needed to reflect current practice. Readers may submit comments to osd-sea@mail.mil | Attn: DEM&S SysML.

"Technical Report: SysML v1 to SysML v2 Model Conversion Approach"

Office of Systems Engineering and Architecture
Office of the Under Secretary of Defense for Research and Engineering
3030 Defense Pentagon
Washington, DC 20301
https://www.cto.mil/sea/
osd-sea@mail.mil | Attn: DEM&S SysML

Contents

Figures

1 Introduction

Systems Modeling Language (SysML) Version 2 (v2) is the next generation of SysML, a standard modeling language used in systems engineering throughout industry and government. The Object Management Group (OMG), an international consortium, adopted SysML v1 in 2007 and is expected to adopt SysML v2 in 2024. SysML v2 supports the evolving practice of model-based systems engineering (MBSE) to address challenges in increasing system complexity and rapid technology change. SysML v2 is intended to increase the effectiveness and adoption of MBSE by improving the language precision, expressiveness, regularity, and interoperability.

The Department of Defense (DoD) uses SysML in system development, and some acquisition programs will need to convert their models from to SysML v1 to SysML v2 while other acquisition program will continue to model in SysML v1. The DoD Office of the Under Secretary of Defense for Research and Engineering (OUSD(R&E)) Systems Engineering and Architecture (SE&A), Digital Engineering, Modeling and Simulation (DEM&S), sponsored a project to develop recommendation for an organization to use to make change programs from SysML v1 to SysML v2. The information is broad in addressing modeling practices, tools, and training but also includes details on how to convert a model from SysML v1 to v2. This report is intended for DoD organizations and programs that are using or planning to use MBSE practices with SysML. This information is offered for reference and is not intended as official DoD policy.

1.1 New in SysML v2

SysML v2 incorporates a new metamodel designed to address system modeling needs while further leveraging the capabilities of the Unified Modeling Language (UML) metamodel upon which SysML v1 is based. SysML v2 includes a textual notation in addition to a graphical notation to increase the language precision. It includes a standard application programming interface (API)[1] to enable interoperability between the system model in SysML v2 and other models and tools that are part of the digital engineering ecosystem.

SysML v2 offers the potential to increase quality by allowing greater visibility into a system beginning early in development and greater precision in design throughout the life cycle. It promotes productivity, agility, and reduced cycle-time by increasing the opportunity for reuse and enhancing interoperability with other engineering disciplines and processes during development.

[1] An application programming interface (API) is “a set of protocols, routines, and tools for building software applications. APIs define how different software components should interact with each other, allowing developers to create applications that can leverage the functionality of other software systems” (SE&A Software Engineering, https://www.cto.mil/sea/swe).

1.2 SysML v1 to SysML v2 Transition

The OMG approved the SysML v2 beta specifications in June 2023[2]. The specifications are now in their finalization phase, during which time tool vendors are asked to provide feedback regarding the specification. The specification is expected to be submitted for final adoption in 2024. Commercial tools are anticipated to become available beginning shortly after final adoption.

Organizations should develop a transition strategy and plan to fully benefit from SysML v2. They should form a transition team responsible for developing and implementing the strategy and plan as part of their existing improvement efforts such as those for digital engineering and MBSE. The strategy and plan should focus on establishing modeling practices, tools, and training and should include pilot projects to assess the impact of proposed updates. The transition team should assess when programs should make the transition and consider potential near, intermediate, and long-term benefits versus the costs and risks of transition. The transition approach should include support for both SysML v1 and v2 models to coexist for some years to come.

The organization should provide the needed support to assist programs in their transition planning and tool acquisition. This includes training team members in the language, methods, and tools, and providing programs with the information and tools established during their pilot phase, including reference models, patterns, and reuse libraries. The organization should continue to provide ongoing subject matter expertise to guide the program through the transition.

Transition planning to SysML v2 should begin before the start of a new program or before a major system upgrade. A smooth transition requires considerable preparation to ensure the program's systems engineering team is properly trained in SysML v2 and has access to the appropriate tools and resources. The project team also developed a Transition Plan Template[3] to assist organizations in planning.

[2] See Object Management Group (OMG) website, https://www.omg.org/spec/SysML/2.0/Beta1.
[3] SysML v1 to SysML v2 Transition Community, SysML v1 to SysML v2 Transition Plan Template https://www.omgwiki.org/MBSE/doku.php?id=mbse:sysml_v2_transition:sysml_v1_to_sysml_v2_transition_guidance

2 SysML v1 to SysML v2 Model Conversion Process

Figure 1 shows the steps in the conversion process from a SysML v1 model to a SysML v2 model which includes: (1) pre-process the SysML v1 model to prepare it for the transformation, (2) transform the SysML v1 model to a SysML v2 model, (3) post-process the SysML v2 model to better leverage the SysML v2 capabilities, and (4) validate that the SysML v2 model accurately reflects the intent of the SysML v1 model.

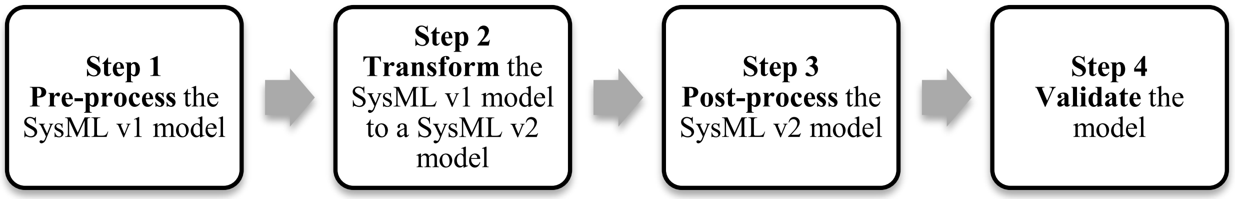

Step 1 **Pre-process** the SysML v1 model
Step 2 **Transform** the SysML v1 model to a SysML v2 model
Step 3 **Post-process** the SysML v2 model
Step 4 **Validate** the model

Figure 1. Model Conversion Process

In addition, further steps may be required to assess the impact of the SysML v2 model on existing artifacts that were derived from the SysML v1 model. The derived artifacts may need to be updated for the SysML v2 effort, but this is considered outside of the scope of the SysML v1 to SysML v2 model conversion. Each of these steps is summarized below.

2.1 Pre-Process

This step involves pre-processing the SysML v1 model to prepare the model for transformation. The required pre-processing will depend on the transformation capability that the modeling tool provides, so it is important to understand the tool capability and limitations. Performing the standard SysML v1 to SysML v2 model transformation requires that the SysML v1 model conform to the SysML v1 specification, so the pre-processing should ensure that SysML v1 model conforms to its specification. Any tool-specific extensions along with other tool customizations to the model may need to be removed. However, the use of stereotypes and profiles are expected to be supported by the transformation.

Certain features of SysML v1, such as adjunct properties, are not incorporated in SysML v2. Part of the pre-processing could be to remove these elements or assess the impact of the transformation on these features and note that they may need to be addressed in the post-processing step.

Circular dependencies should be identified to determine if and how they may impact the transformation and addressed accordingly. The SysML v1 model may also need to be reorganized to enable an incremental conversion process.

Creating a well-formed SysML v1 model that conforms to good practice will facilitate the conversion process. Model validation errors should be resolved to ensure the model is well-formed.

Standard modeling conventions should be applied such as consistent naming conventions and ambiguities and redundancies in the model should be minimized.

2.2 Transform

This step involves transforming the pre-processed SysML v1 model to a SysML v2 model. A SysML v1 model can be transformed to a SysML v2 model using a tool that can execute the standard SysML v1 to SysML v2 transformation specification. The standard transformation requires that the SysML v1 model be conformant to the SysML v1 specification to be transformed to a conformant SysML v2 model.

The SysML v1 to v2 transformation specification defines the rules for transforming each kind of element in SysML v1 to a corresponding element in SysML v2. The transformation also includes rules for cases where there is no corresponding SysML v2 element. For example, a block in SysML v1 includes a meta property called '*isEncapsulated*'. There is no equivalent concept in SysML v2 since the SysML v2 language designers did not see a need for this. However, there is a rule for how to address this in the transformation.

The tool should generate validation errors and warnings to indicate what aspects of the transformation were not successful. In addition, a manual inspection should be performed to compare the SysML v2 model with the SysML v1 model.

2.3 Post-Process

This step involves post-processing the SysML v2 model to leverage the SysML v2 capabilities. The transformed SysML v2 model may need to be reorganized and refactored to fully leverage the SysML v2 capabilities. The reorganizing and refactoring should apply the usage-focused modeling paradigm which is briefly discussed in the section entitled “Post-process the SysML v2 Skyzer model.”

2.4 Validate

It is imperative to validate that the SysML v2 model accurately reflects the intent of the SysML v1 model. This can be done by comparing the two models. This may include reproducing selected views of the SysML v2 model such as a system hierarchy and carefully comparing it with the system hierarchy in the SysML v1 model. It is anticipated that tool vendors may be able to generate automated comparison reports to assist in the inspection. Comparing execution and analysis results from the SysML v2 model with the corresponding execution and analysis results from the SysML v1 model may also assist in the validation. (Note: SysML v2 execution semantics are still being specified as of the date of this writing).

A tool is expected to support the SysML v2 standard views which can render similar information that is contained in the nine standard SysML v1 diagrams. However, the layout information is not preserved and may need to be adjusted manually to align with the original SysML v1 diagram.

3 Other Considerations in the Model Conversion Process

3.1 Whether to Convert

Before making the conversion, a project may want to evaluate the cost of converting a SysML v1 model to a SysML v2 model versus the cost of developing a new SysML v2 model from scratch. It may be more cost-effective to start from scratch if the SysML v1 model was not maintained, or if its scope of the SysML v1 model is not consistent with the current effort.

3.2 Incremental Model Conversion

Projects should perform the conversion process incrementally rather than as a one-time process. As part of the pre-processing, the SysML v1 model can be partitioned to reduce the coupling between the parts of the model that will be incrementally transformed. For example, the model can be partitioned into packages that contain the structure, behavior, and requirements and further partitioned into mission, system, and subsystem levels. The incremental conversion process may first transform the structure, then transform the behavior, and then transform the requirements.

3.3 One-Way Transformation

The transformation occurs in a one-way direction from SysML v1 to SysML v2. There is no standard to transform a SysML v2 model to a SysML v1 model because many of the capabilities in SysML v2 are not supported in SysML v1. For example, SysML v1 supports a block decomposition but does not support a SysML v2 part decomposition.

3.4 Classified Models

The transformation of a classified SysML v1 model should preserve all classification markings in the SysML v2 model. A standard security extension should be applied that leverages the metadata capability in SysML v2. A project should define a process to ensure all markings are properly applied and includes manual inspection of the model. The same classification procedures that apply to the SysML v1 model should apply to the SysML v2 model.

3.5 Configuration Management

Projects can apply the SysML v2 API configuration management services to the SysML v2 models beginning with the initial transformation. Typical branch and merge concepts can be used to manage updates to the model. The configuration management of the SysML v2 model should be incorporated into the broader life cycle management environment using workflow or issue management applications such as Jira.

4 Example SysML v1 to SysML v2 Conversion

To investigate the conversion of an existing SysML v1 model to SysML v2, DEM&S used the Skyzer Mission Model, a fictional unmanned aerial vehicle (UAV) model developed by the Systems Engineering Research Center (SERC) under contract with the Naval Air (NAVAIR) Systems Command. SERC developed a series of models to represent the mission and system design and to demonstrate the application of typical modeling practices and methods. [4]

The Skyzer Mission Model portrays a UAV launched from a ship to perform a search and rescue mission. The mission model consists of approximately 5,300 model elements. It includes 6 of the 9 standard SysML v1 diagrams, and requirements tables, but it does not include any state machine, parametric, or requirements diagrams as shown in Figure 2.

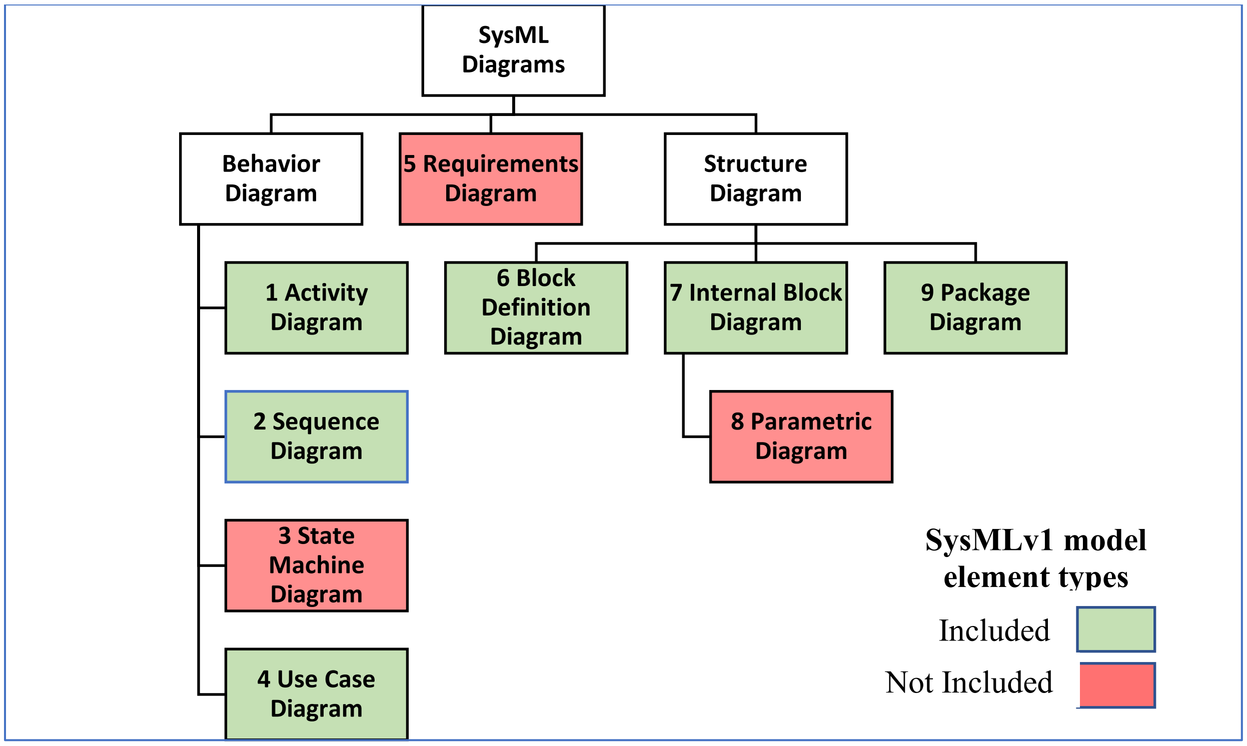


Figure 2. SysML v1 Diagram Types Used in the Skyzer Mission Model

The model includes many kinds of SysML v1 model elements including packages, dependencies, blocks, attributes, parts, connections, associations, use cases, actors, activities, actions, swim lanes, control flows, object flows, lifelines, messages, requirements, constraints, and trace and satisfy relationships. The model also includes some stereotypes to create language extensions and some customizations that are unique to the tool such as a glossary, acronym list, a legend, and some custom images. Some of the more common elements that are not included in this model are proxy

[4] The original SysML v1 Skyzer Mission Model is available publicly at the following link. To open and view the file, enter User: openmbeeguest and Password: guest. The tool used to create this model was MagicDraw version 19.0 sp4.

and full ports, interface blocks, states, transitions, test cases, derive and verify relationships, constraint blocks, constraint properties, binding connections, views, and viewpoints.

The SysML v2 modeling environment is the pilot implementation that was developed as part of the SysML v2 Submission Team (SST) to validate the SysML v2 language (Seidewitz & Bajaj). This implementation is integrated into the Jupyter Lab environment to provide support for creating SysML v2 models using the textual notation. The graphical visualization environment is adapted from the open-source PlantUML application that also was integrated into the Jupyter environment. The modified PlantUML visualization capability is limited and is not entirely conformant to the SysML v2 specifications. It is anticipated that the visualization capability will be substantially improved when commercial tools become available.

The SysML v2 model was created using the Jupyter environment and is available in two formats using the Jupyter extension. jpynb, and a SysML extension that can be opened in most text editors. The link to a publicly available site can be used to experiment with the SysML v2 textual notation.

Since the objective for converting this example was to illustrate the model conversion approach, only representative parts of the model were converted and not the entire model. The conversion process was performed manually since automation was not available for this effort. The focus for this example was on the transformation and post-processing steps. The manual transformation precluded the need to pre-process the model. The transformation and post-processing steps are described below.

4.1 Transform SysML v1 Skyzer Model to SysML v2

An implementation of the transformation specification is not currently available. The transformation was performed by manually creating SysML v2 elements in the Jupyter environment that corresponded to elements in the SysML v1 model. Significant portions of the SysML v1 model were transformed to demonstrate the approach.

The mapping from SysML v1 elements to SysML v2 elements was based on the modelers experience with SysML v2 rather than following the strict rules defined in the SysML v1 to SysML v2 transformation specification. Much of the mapping is straightforward, such as a block in SysML v1 is transformed to a part def in SysML v2 and a requirement in SysML v1 is transformed to a requirement def in SysML v2. There will be differences between the results of the manual transformation and the results of an automated transformation. However, the manual transformation should be a reasonable approximation of the expected results from an automated transformation after pre-processing.

The steps to transform a SysML v1 model to a SysML v2 model are performed incrementally as follows:

- Transform Package Structure
- Transform Blocks and their Parts
- Transform Ports and Connectors
- Transform Value Properties and Value Types
- Transform Requirements and their Hierarchy
- Transform Use Cases
- Transform Activities
- Transform Interactions (e.g., sequence diagrams)
- Transform State-Based Behavior
- Transform Parametrics
- Transform Requirements Relationships
- Transform Other Elements
- Transform Stereotypes
- Transform Customizations

4.1.1 Transform Package Structure

The first step in the transformation process was to transform the SysML v1 Skyzer Mission Model package structure shown in the package diagram in **Figure 3**.

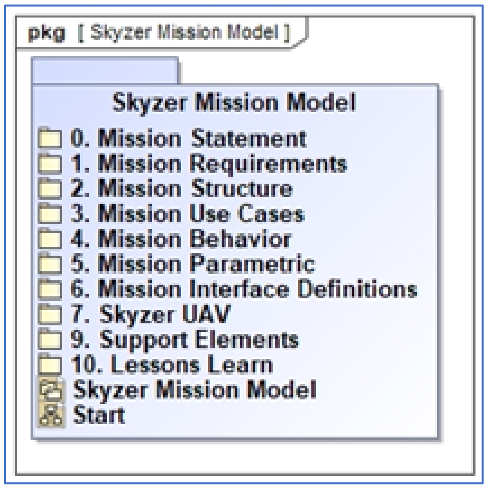
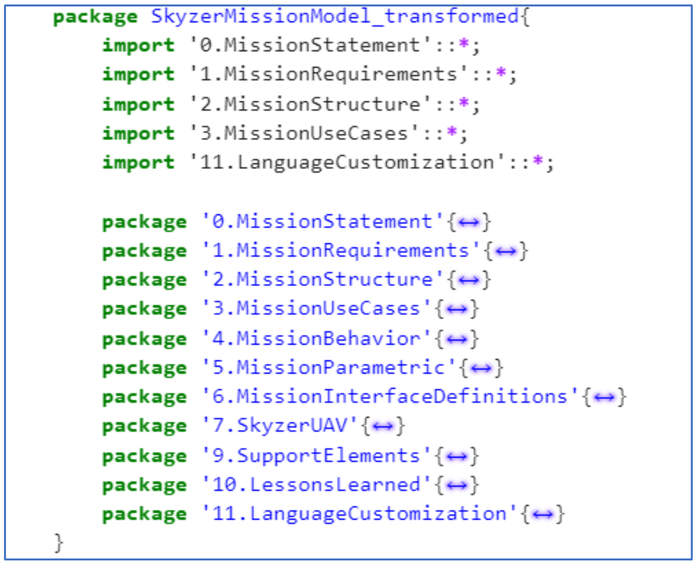


Figure 3. Skyzer Mission Model Package Structure in SysML v1

The corresponding SysML v2 package structure is shown below in **Figure 4**. The package names must be included in quotes because the name begins with a number. The import statements are included to make the contents of selected packages visible to the rest of the model.



```
package SkyzerMissionModel_transformed{
    import '0.MissionStatement'::*;
    import '1.MissionRequirements'::*;
    import '2.MissionStructure'::*;
    import '3.MissionUseCases'::*;
    import '11.LanguageCustomization'::*;

    package '0.MissionStatement'{↔}
    package '1.MissionRequirements'{↔}
    package '2.MissionStructure'{↔}
    package '3.MissionUseCases'{↔}
    package '4.MissionBehavior'{↔}
    package '5.MissionParametric'{↔}
    package '6.MissionInterfaceDefinitions'{↔}
    package '7.SkyzerUAV'{↔}
    package '9.SupportElements'{↔}
    package '10.LessonsLearned'{↔}
    package '11.LanguageCustomization'{↔}
}
```

Figure 4. Skyzer Mission Model Package Structure in SysML v2

4.1.2 Transform Blocks and their Parts

The next step in the transformation was to transform the blocks and their parts that are generally depicted on a block definition diagram. The blocks in the SysML v1 model are distributed across many packages including the Mission Structure, Mission Use Cases, Mission Parametric, Skyzer UAV, and Support Elements.

Figures 5 and 6 show some of the blocks and their structure in the OV-1 and the Skyzer Mission Domain model respectively. The blocks in the OV-1 are the same blocks that are shown in the Mission Domain Model. In the OV-1, the blocks are related through dependencies (dashed lines with arrowheads). The symbols on the dependency relationships are defined in the legend as graphical adornments that have no relationship to other model elements. The legend is an example of a tool customization.

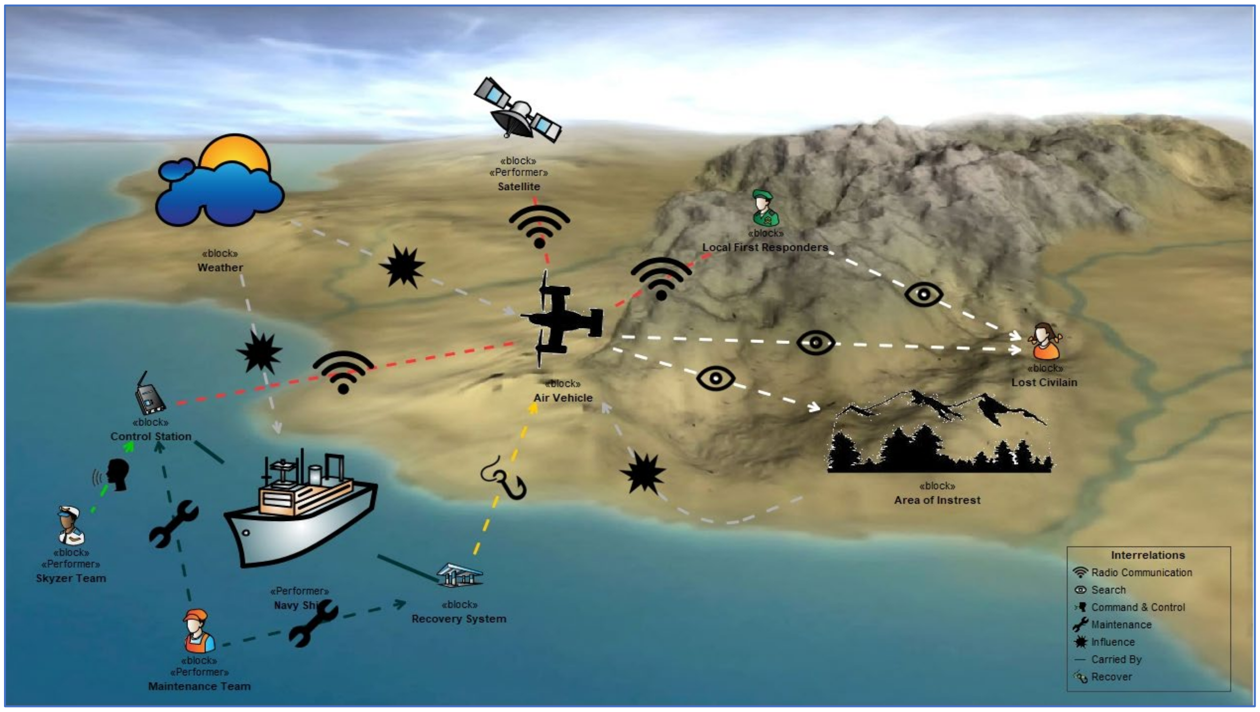


Figure 5. Skyzer OV-1 in SysML v1

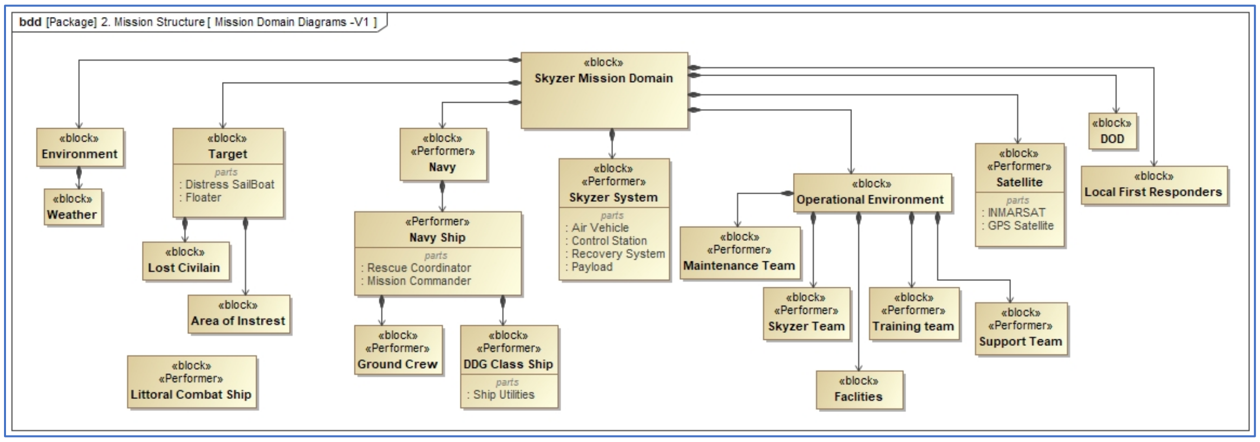

bdd [Package] 2. Mission Structure [Mission Domain Diagrams -V1]
«block» Skyzer Mission Domain
«block» Environment
«block» Weather
«block» Target
parts
: Distress SailBoat
: Floater
«block» Lost Civilain
«block» Area of Instrest
«block» «Performer» Littoral Combat Ship
«block» «Performer» Navy
«Performer» Navy Ship
parts
: Rescue Coordinator
: Mission Commander
«block» «Performer» Ground Crew
«block» «Performer» DDG Class Ship
parts
: Ship Utilities
«block» «Performer» Skyzer System
parts
: Air Vehicle
: Control Station
: Recovery System
: Payload
«block» «Performer» Maintenance Team
«block» Operational Environment
«block» «Performer» Skyzer Team
«block» «Performer» Training team
«block» Faclities
«block» «Performer» Support Team
«block» «Performer» Satellite
parts
: INMARSAT
: GPS Satellite
«block» DOD
«block» Local First Responders

Figure 6. Skyzer Mission Domain Block Definition Diagram in SysML v1

Figure 7 shows the OV-1 in SysML v2 corresponding to the OV-1 in SysML v1. The blocks in SysML v1 map to part definitions in SysML v2 and the dependencies in SysML v1 map to dependencies in SysML v2. The legend in the SysML v1 model is a custom feature that has no standard mapping to SysML v2 and would require a custom mapping.

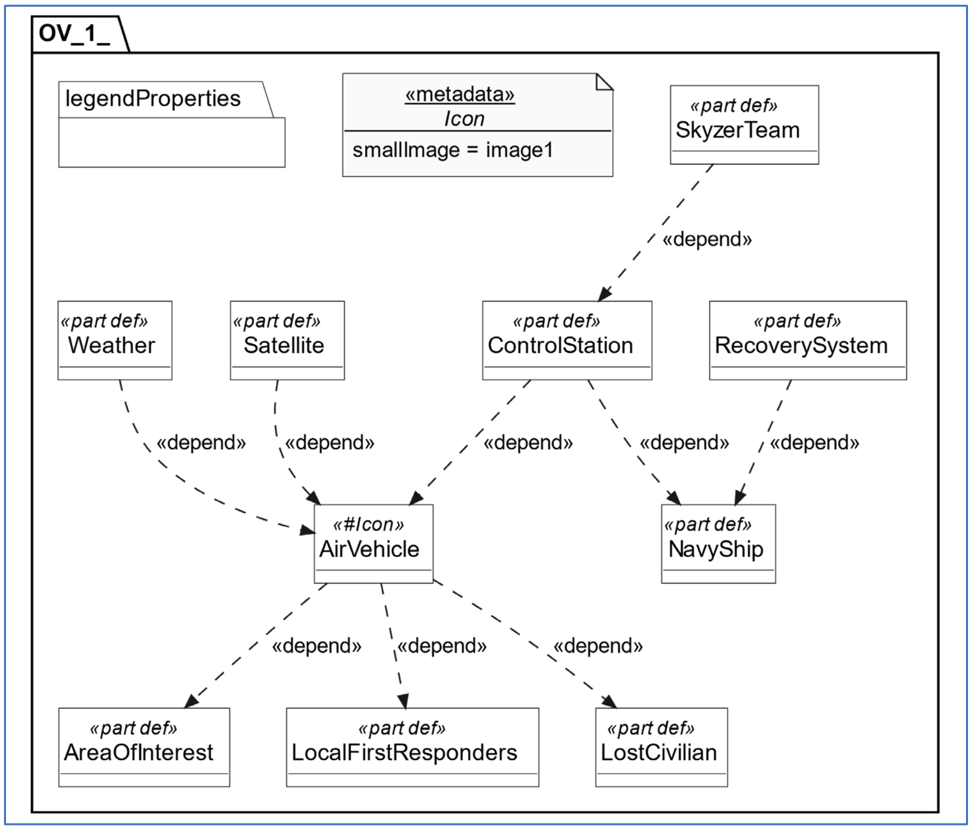


Figure 7. Skyzer OV-1 in SysML v2

Figure 8 shows the SysML v2 structure that corresponds to the SysML v1 Skyzer Mission Model block structure in **Figure 6**. A portion of the SysML v2 diagram is expanded in **Figure 9**. The blocks in SysML v1 are transformed to part definitions in SysML v2 and the part properties in SysML v1 are transformed to parts in SysML v2. The composite associations in SysML v1 are mapped to feature memberships in SysML v2 with a black diamond on the owning end of the relationship. The line with the arrowhead and the two dots is a "defined by" relationship between a part and a part definition.

The SysML v2 structure corresponds to the same structure as shown in SysML v1. The structure shows that a part definition is composed of parts that are defined by part definitions that are composed of other parts. This pattern applies recursively down the system hierarchy. The post-processed model can simplify this structure considerably using the usage focused modeling paradigm to represent the structure as a part hierarchy without having to traverse from part def to part to part def to part.

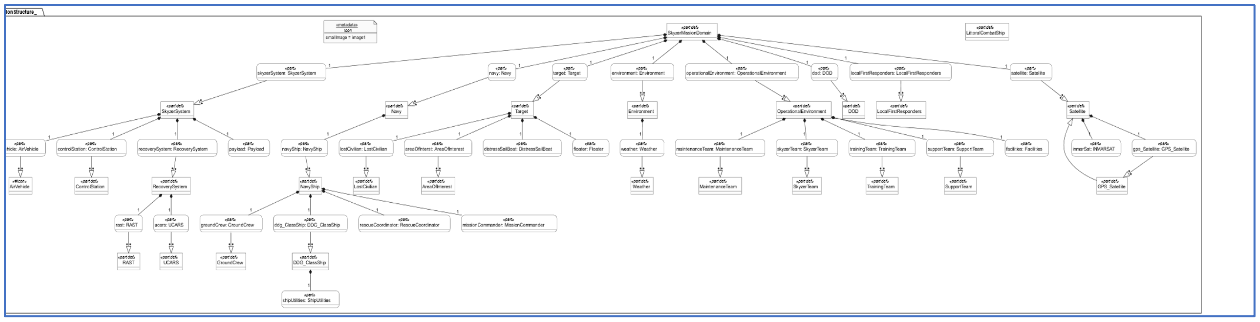


Figure 8. Skyzer Mission Domain Part Def Structure in SysML v2

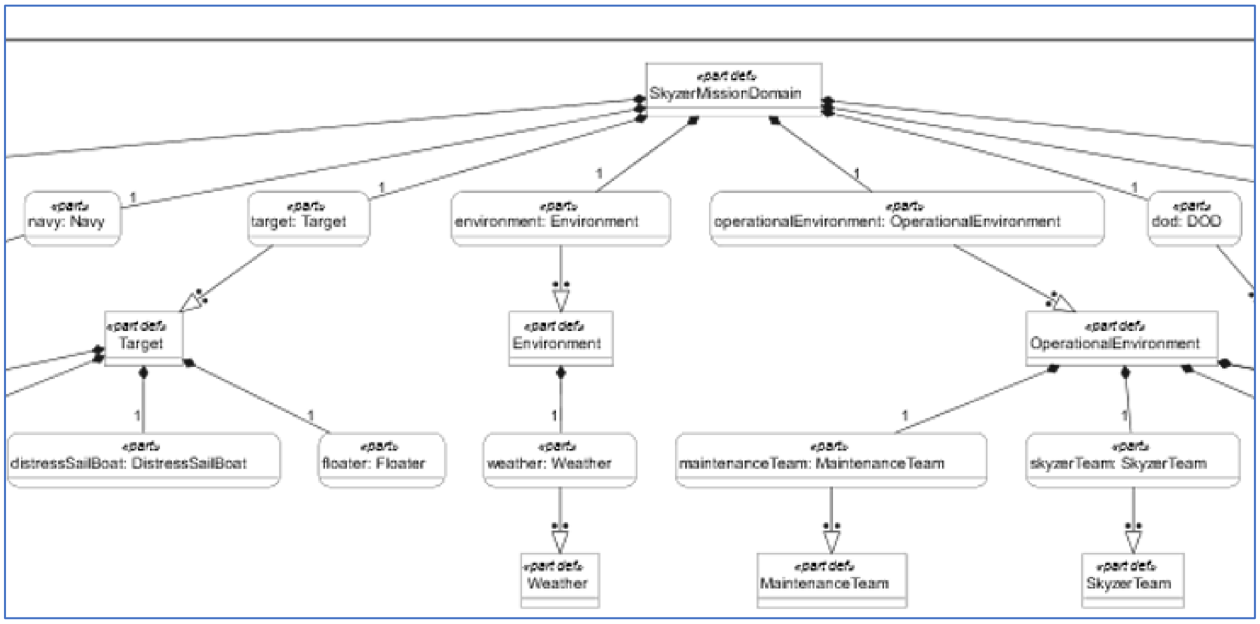


Figure 9. Skyzer Mission Domain Part Def Structure in SysML v2

4.1.3 Transform Ports and Connectors

The next step was to transform SysML v1 interconnections that are depicted on internal block diagrams.

The Skyzer System black box interfaces in the SysML v1 model are shown in **Figure 10** on an internal block diagram. The SysML v1 model did not include the use of ports on parts. The information items that flow across the connectors are contained in the Mission Interface Definitions package.

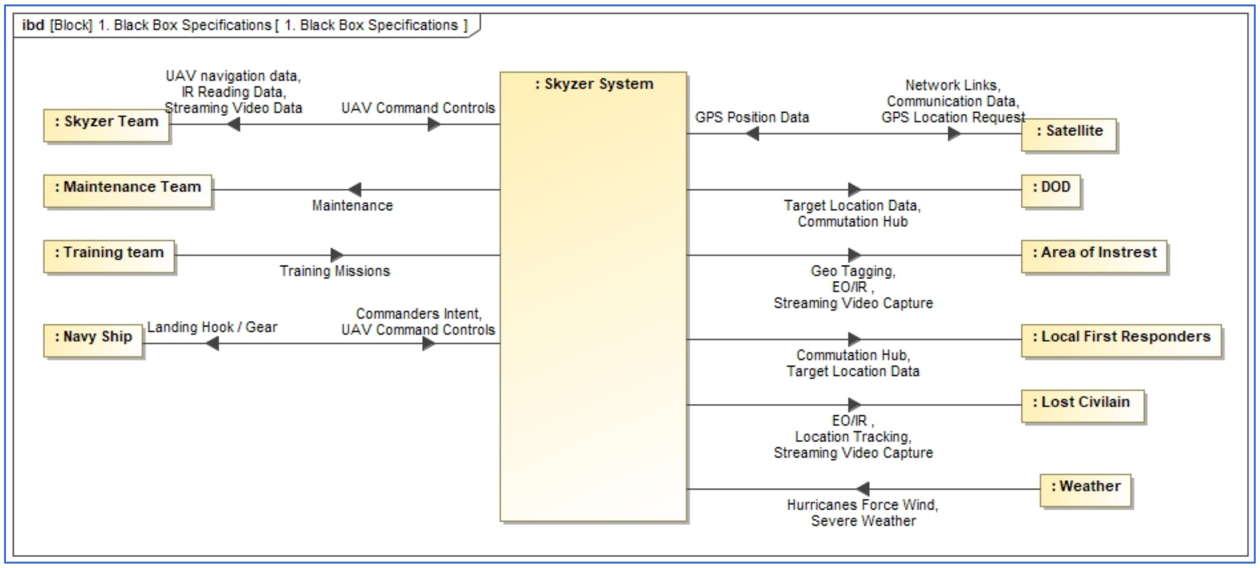
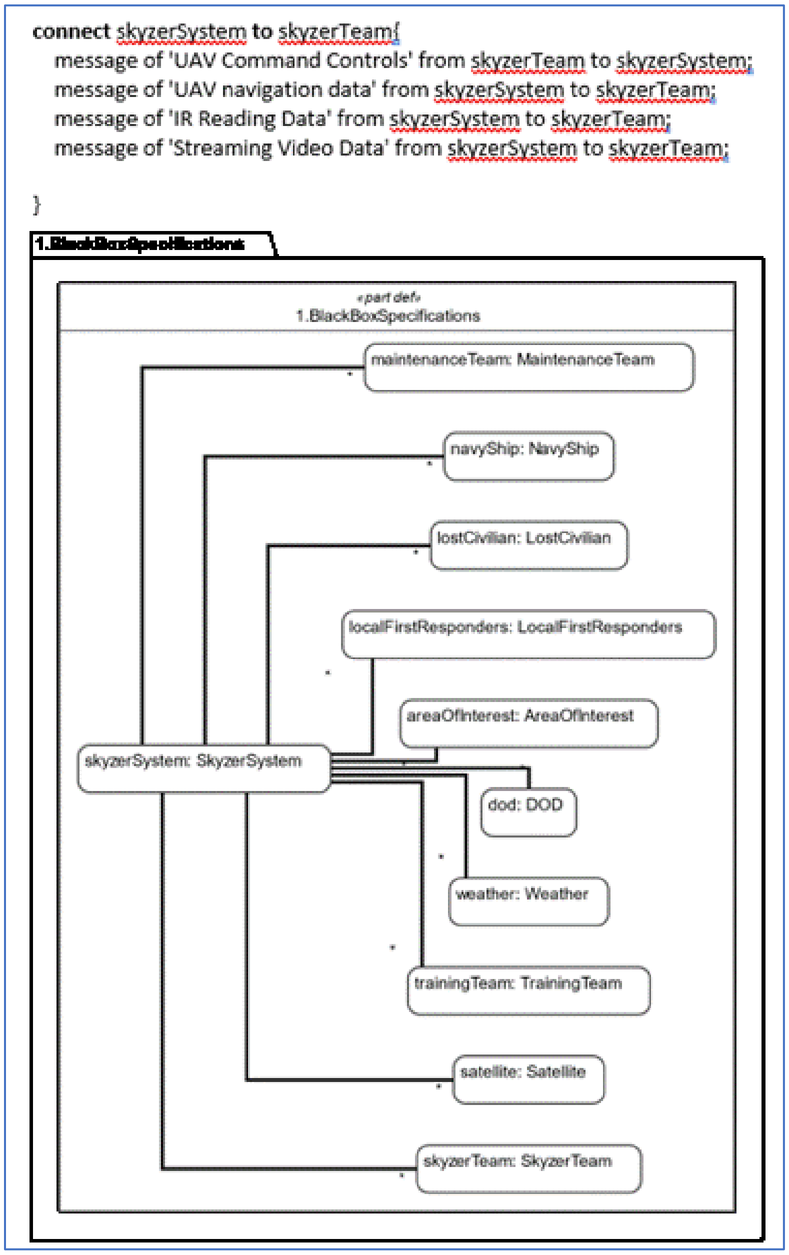

ibd [Block] 1. Black Box Specifications [1. Black Box Specifications]
: Skyzer System
: Skyzer Team
UAV navigation data, IR Reading Data, Streaming Video Data
UAV Command Controls
: Maintenance Team
Maintenance
: Training team
Training Missions
: Navy Ship
Landing Hook / Gear
Commanders Intent, UAV Command Controls
GPS Position Data
Network Links, Communication Data, GPS Location Request
: Satellite
: DOD
Target Location Data, Commutation Hub
: Area of Instrest
Geo Tagging, EO/IR , Streaming Video Capture
: Local First Responders
Commutation Hub, Target Location Data
: Lost Civilain
EO/IR , Location Tracking, Streaming Video Capture
: Weather
Hurricanes Force Wind, Severe Weather

Figure 10. Skyzer System Black Box Interfaces in SysML v1

A SysML v1 connector transforms to a SysML v2 connection. A SysML v1 proxy port and interface block transform to a SysML v2 port and port definition, respectively. The items that flow across the connectors are mapped to item definitions.

The corresponding SysML v2 interconnection diagram between the Skyzer System and the external elements is shown in **Figure 11**; however, the items that flow are not displayed due to limitations of the current pilot implementation visualization. The items that flow are represented as item definitions and can be seen in the SysML v2 textual model fragment below. This fragment shows the items that flow between the s*kyzerTeam* and the *skyzerSystem,* which are represented as messages contained by the connection.



```
connect skyzerSystem to skyzerTeam{
   message of 'UAV Command Controls' from skyzerTeam to skyzerSystem;
   message of 'UAV navigation data' from skyzerSystem to skyzerTeam;
   message of 'IR Reading Data' from skyzerSystem to skyzerTeam;
   message of 'Streaming Video Data' from skyzerSystem to skyzerTeam;

}
```

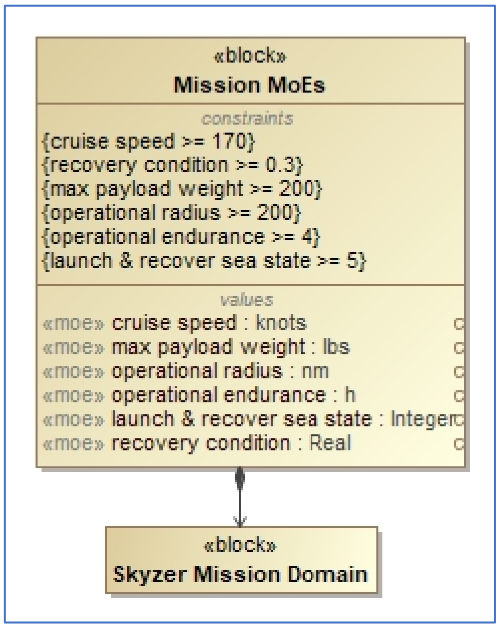
1.BlackBoxSpecifications
«part def»
1.BlackBoxSpecifications
maintenanceTeam: MaintenanceTeam
navyShip: NavyShip
lostCivilian: LostCivilian
localFirstResponders: LocalFirstResponders
areaOfInterest: AreaOfInterest
skyzerSystem: SkyzerSystem
dod: DOD
weather: Weather
trainingTeam: TrainingTeam
satellite: Satellite
skyzerTeam: SkyzerTeam

Items that flow are not displayed.

Figure 11. Skyzer System Black Box Interconnection in SysML v2

4.1.4 Transform Value Properties and Value Types

Figure 12 shows the Skyzer measures of effectiveness as value properties with the moe stereotype applied. Each value property is typed by a value type that is intended to include a quantity kind and unit such as knots. The value properties are constrained as shown in the constraints compartment.


«block»
Mission MoEs
constraints
{cruise speed >= 170}
{recovery condition >= 0.3}
{max payload weight >= 200}
{operational radius >= 200}
{operational endurance >= 4}
{launch & recover sea state >= 5}
values
«moe» cruise speed : knots
«moe» max payload weight : lbs
«moe» operational radius : nm
«moe» operational endurance : h
«moe» launch & recover sea state : Integer
«moe» recovery condition : Real
«block»
Skyzer Mission Domain

Constraints are imposed on the value properties.

Figure 12. Skyzer MoEs in SysML v1 as Value Properties and Value Types with Units.

The corresponding SysML v2 measures of effectiveness and constraints are shown in **Figure 13**. The SysML v1 value properties and value types transform to SysML v2 attributes and attribute definitions, and SysML v1 constraints transform to SysML v2 constraints. The SysML v1 moe stereotype is transformed to a standard extension that is defined in the SysML v2 model library called the *ParametersOfInterestMetadata* library. The moe key word is not displayed in **Figure 13** due to limitations of the visualization, but can be seen in the textual notation below in **Figure 14** designated as #moe:

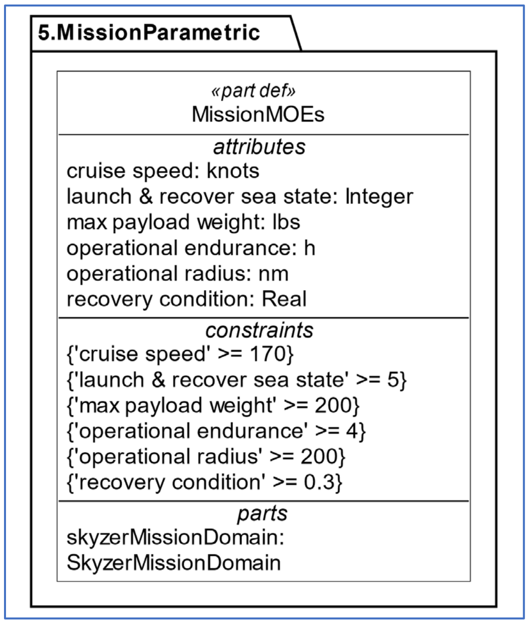
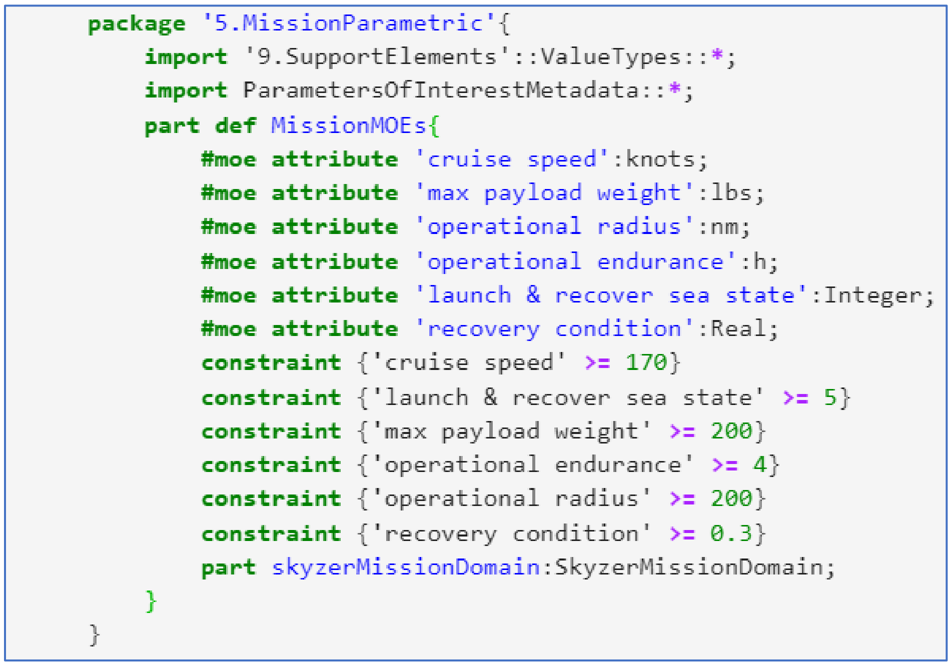


Figure 13. Skyzer MoEs in SysML v2 as Attributes and Attribute Definitions with Units and Constraints

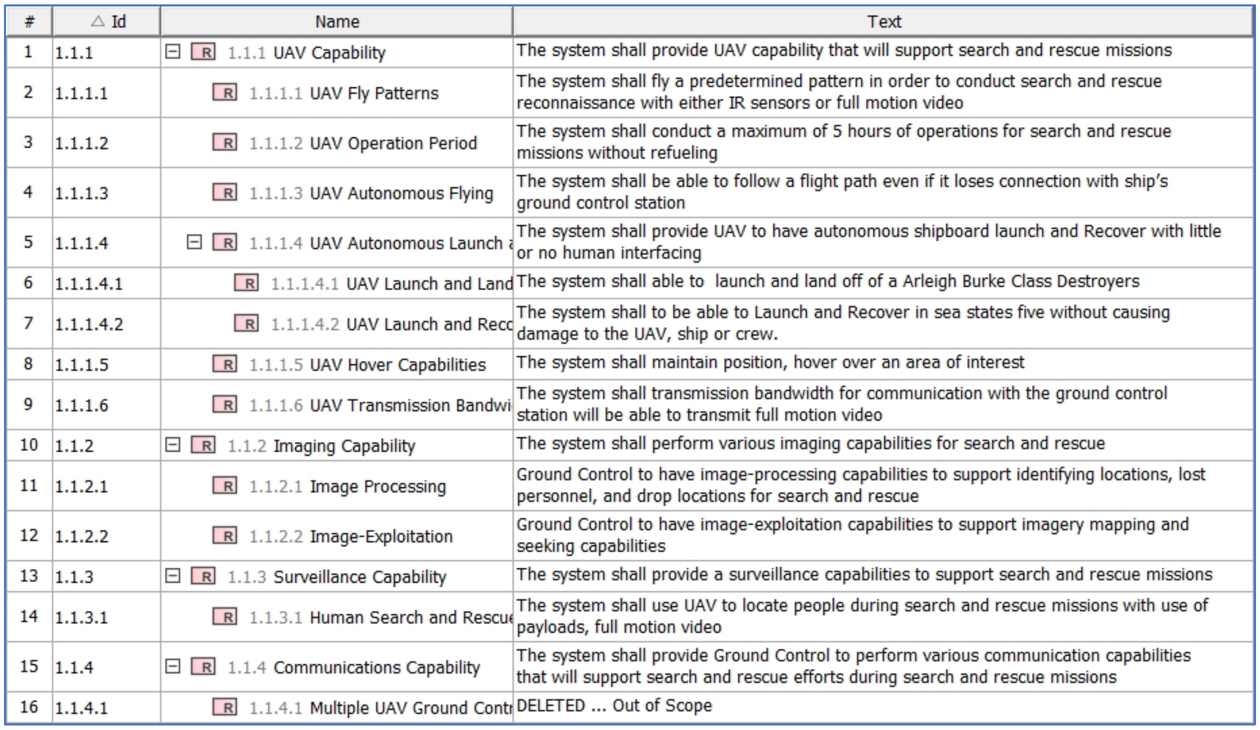


```
package '5.MissionParametric'{
    import '9.SupportElements'::ValueTypes::*;
    import ParametersOfInterestMetadata::*;
    part def MissionMOEs{
        #moe attribute 'cruise speed':knots;
        #moe attribute 'max payload weight':lbs;
        #moe attribute 'operational radius':nm;
        #moe attribute 'operational endurance':h;
        #moe attribute 'launch & recover sea state':Integer;
        #moe attribute 'recovery condition':Real;
        constraint {'cruise speed' >= 170}
        constraint {'launch & recover sea state' >= 5}
        constraint {'max payload weight' >= 200}
        constraint {'operational endurance' >= 4}
        constraint {'operational radius' >= 200}
        constraint {'recovery condition' >= 0.3}
        part skyzerMissionDomain:SkyzerMissionDomain;
    }
}
```

Figure 14. Skyzer Measures of Effectiveness (MOEs) in SysML v2 Textual Notation

4.1.5 Transform Requirements and their Hierarchy

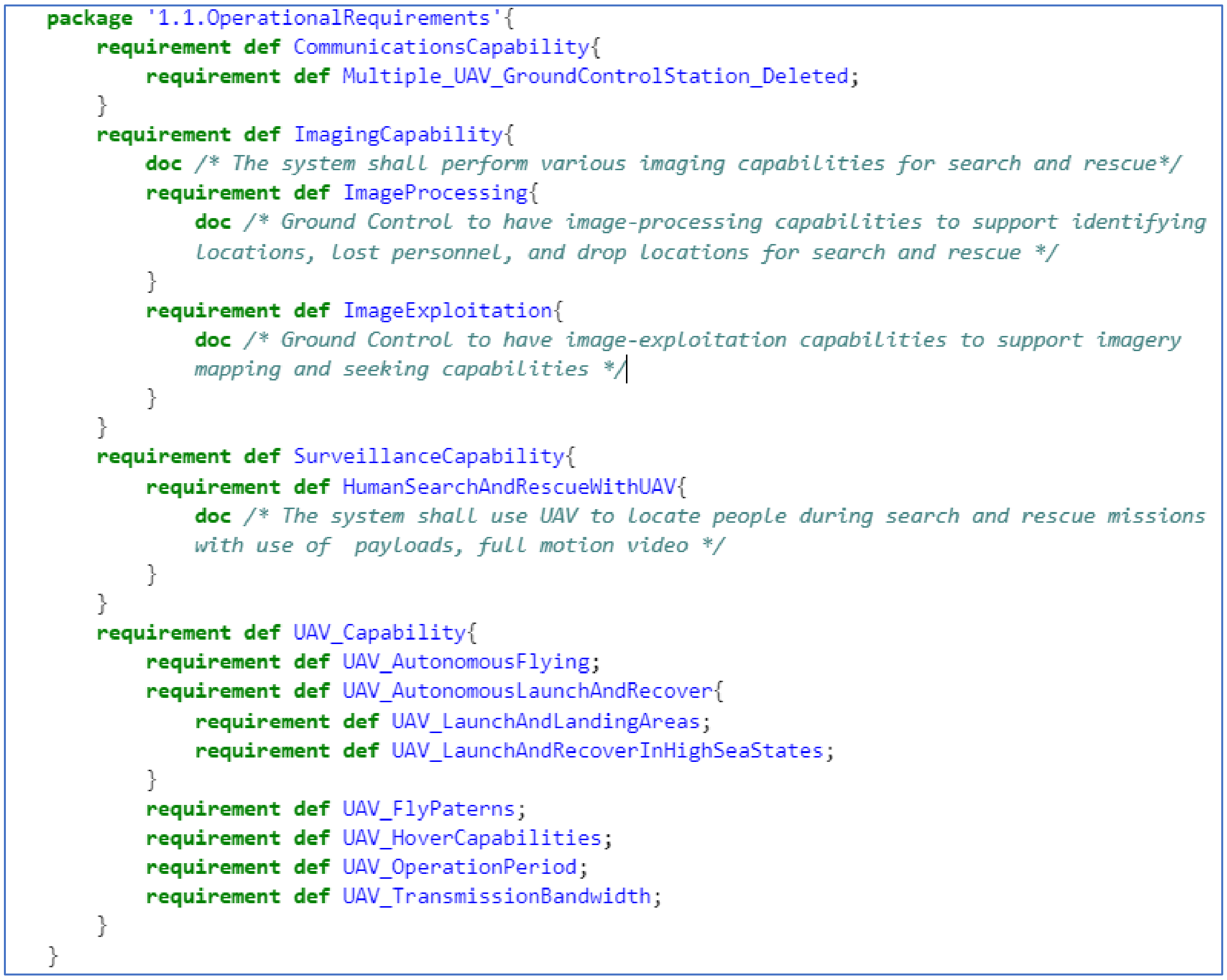
The next step in the transformation was to transform the requirements and their containment hierarchy. In SysML v1, requirements are generally depicted using requirements diagrams and requirements tables. The requirements in the Skyzer Mission Model were contained in the Mission Requirements package and depicted using requirements tables. The SysML v1 requirements table for the Operational Requirements is shown in **Figure 15**.



#	△ Id	Name	Text
1	1.1.1	⊟ R 1.1.1 UAV Capability	The system shall provide UAV capability that will support search and rescue missions
2	1.1.1.1	R 1.1.1.1 UAV Fly Patterns	The system shall fly a predetermined pattern in order to conduct search and rescue reconnaissance with either IR sensors or full motion video
3	1.1.1.2	R 1.1.1.2 UAV Operation Period	The system shall conduct a maximum of 5 hours of operations for search and rescue missions without refueling
4	1.1.1.3	R 1.1.1.3 UAV Autonomous Flying	The system shall be able to follow a flight path even if it loses connection with ship's ground control station
5	1.1.1.4	⊟ R 1.1.1.4 UAV Autonomous Launch a	The system shall provide UAV to have autonomous shipboard launch and Recover with little or no human interfacing
6	1.1.1.4.1	R 1.1.1.4.1 UAV Launch and Land	The system shall able to launch and land off of a Arleigh Burke Class Destroyers
7	1.1.1.4.2	R 1.1.1.4.2 UAV Launch and Reco	The system shall to be able to Launch and Recover in sea states five without causing damage to the UAV, ship or crew.
8	1.1.1.5	R 1.1.1.5 UAV Hover Capabilities	The system shall maintain position, hover over an area of interest
9	1.1.1.6	R 1.1.1.6 UAV Transmission Bandwi	The system shall transmission bandwidth for communication with the ground control station will be able to transmit full motion video
10	1.1.2	⊟ R 1.1.2 Imaging Capability	The system shall perform various imaging capabilities for search and rescue
11	1.1.2.1	R 1.1.2.1 Image Processing	Ground Control to have image-processing capabilities to support identifying locations, lost personnel, and drop locations for search and rescue
12	1.1.2.2	R 1.1.2.2 Image-Exploitation	Ground Control to have image-exploitation capabilities to support imagery mapping and seeking capabilities
13	1.1.3	⊟ R 1.1.3 Surveillance Capability	The system shall provide a surveillance capabilities to support search and rescue missions
14	1.1.3.1	R 1.1.3.1 Human Search and Rescue	The system shall use UAV to locate people during search and rescue missions with use of payloads, full motion video
15	1.1.4	⊟ R 1.1.4 Communications Capability	The system shall provide Ground Control to perform various communication capabilities that will support search and rescue efforts during search and rescue missions
16	1.1.4.1	R 1.1.4.1 Multiple UAV Ground Contr	DELETED ... Out of Scope

Figure 15. Operational Requirements in SysML v1 Requirements Table

A SysML v1 requirement maps to a SysML v2 requirement definition. The mapping rules in the transformation specification include additional constraints on the usages of the requirement definition. The SysML v2 requirement definitions in **Figure 16** correspond to the SysML v1 Operational Requirements in **Figure 15**. A subset of the requirements includes the shall statements using the **doc** key word.



```
package '1.1.OperationalRequirements'{
    requirement def CommunicationsCapability{
        requirement def Multiple_UAV_GroundControlStation_Deleted;
    }
    requirement def ImagingCapability{
        doc /* The system shall perform various imaging capabilities for search and rescue*/
        requirement def ImageProcessing{
            doc /* Ground Control to have image-processing capabilities to support identifying
            locations, lost personnel, and drop locations for search and rescue */
        }
        requirement def ImageExploitation{
            doc /* Ground Control to have image-exploitation capabilities to support imagery
            mapping and seeking capabilities */
        }
    }
    requirement def SurveillanceCapability{
        requirement def HumanSearchAndRescueWithUAV{
            doc /* The system shall use UAV to locate people during search and rescue missions
            with use of  payloads, full motion video */
        }
    }
    requirement def UAV_Capability{
        requirement def UAV_AutonomousFlying;
        requirement def UAV_AutonomousLaunchAndRecover{
            requirement def UAV_LaunchAndLandingAreas;
            requirement def UAV_LaunchAndRecoverInHighSeaStates;
        }
        requirement def UAV_FlyPaterns;
        requirement def UAV_HoverCapabilities;
        requirement def UAV_OperationPeriod;
        requirement def UAV_TransmissionBandwidth;
    }
}
```

Figure 16. Operational Requirements in SysML v2 Textual Notation

SysML v1 uses containment to represent a requirement hierarchy, which is depicted graphically as a line with a crosshair symbol. Although the containment is depicted graphically, it is not an actual (i.e., reified) relationship in the SysML v1 model. The SysML v1 containment maps to a membership relationship in SysML v2 which is a reified relationship that is explicitly represented in the SysML v2 model. This SysML v2 membership relationship is depicted graphically using the same line and crosshair symbol as SysML v1 containment. A SysML v2 graphical view of some of the Operational Requirements is shown in **Figure 17**.

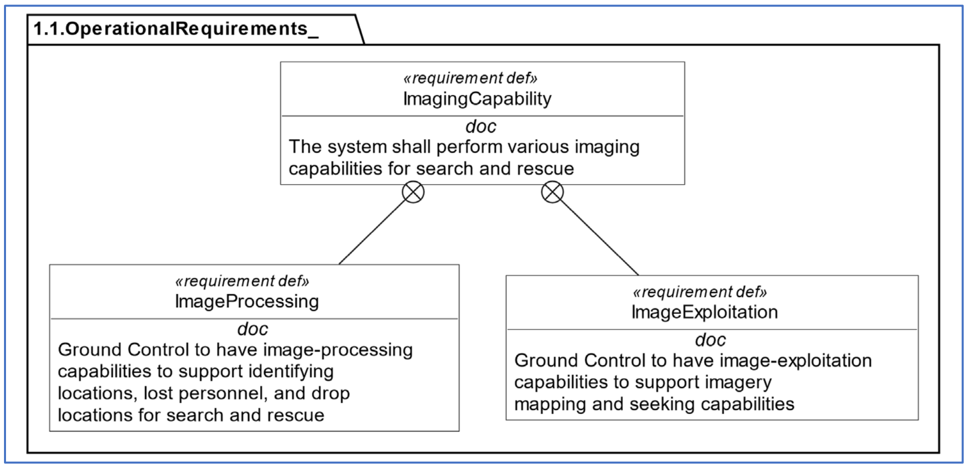

1.1.OperationalRequirements_
«requirement def»
ImagingCapability
doc
The system shall perform various imaging capabilities for search and rescue
«requirement def»
ImageProcessing
doc
Ground Control to have image-processing capabilities to support identifying locations, lost personnel, and drop locations for search and rescue
«requirement def»
ImageExploitation
doc
Ground Control to have image-exploitation capabilities to support imagery mapping and seeking capabilities

Figure 17. Operational Requirements in SysML v2 Graphical Notation (partial)

The requirement id is included in the SysML v1 model but not included in the transformation. SysML v2 provides a mechanism for any element to contain an id using a short name that is contained in brackets. For example, the SysML v1 requirement called Imaging Capability has an id 1.1.1. This requirement can be transformed to the SysML v2 requirement definition in **Figure 17** as <1.1.1> Imaging Capability.

4.1.6 Transform Use Cases

The next step in the transformation was to transform the use cases, which are generally shown in SysML v1 use case diagrams. The use cases in the Skyzer Mission Model are contained in the Mission Use Cases package. The Skyzer Mission Use Cases are shown in **Figure 18**. Some use cases are specializations of other use cases. The use cases are stereotyped extensions of the standard SysML v1 use case. Each use case contains a Use Case Number and several other stereotype properties that are not shown in the diagram. The Skyzer System is expressed as a block and the external systems are expressed as actors with the stereotype stakeholder and Performer. A stakeholder concern is a comment that is related to the stakeholder stereotype.

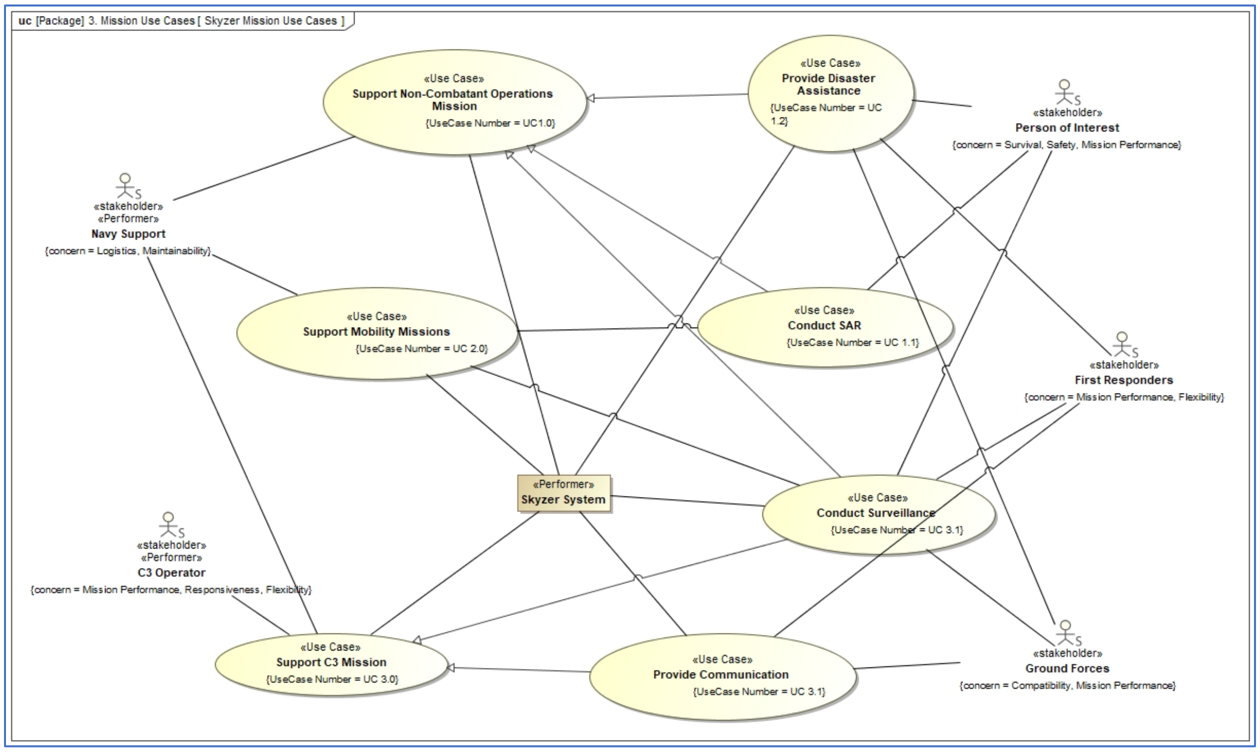

uc [Package] 3. Mission Use Cases [Skyzer Mission Use Cases]
«Use Case» Support Non-Combatant Operations Mission {UseCase Number = UC1.0}
«Use Case» Provide Disaster Assistance {UseCase Number = UC 1.2}
«stakeholder» Person of Interest {concern = Survival, Safety, Mission Performance}
«stakeholder» «Performer» Navy Support {concern = Logistics, Maintainability}
«Use Case» Support Mobility Missions {UseCase Number = UC 2.0}
«Use Case» Conduct SAR {UseCase Number = UC 1.1}
«stakeholder» First Responders {concern = Mission Performance, Flexibility}
«Performer» Skyzer System
«Use Case» Conduct Surveillance {UseCase Number = UC 3.1}
«stakeholder» «Performer» C3 Operator {concern = Mission Performance, Responsiveness, Flexibility}
«Use Case» Support C3 Mission {UseCase Number = UC 3.0}
«Use Case» Provide Communication {UseCase Number = UC 3.1}
«stakeholder» Ground Forces {concern = Compatibility, Mission Performance}

Figure 18. Skyzer Mission Use Cases in SysML v1

A SysML v1 use case maps to a SysML v2 use case definition. The SysML v1 actors map to SysML v2 actors and the SysML v1 use case subject maps to the SysML v2 use case subject. The SysML v1 association between the use case and its actors map to a feature membership between the use case and each of its actors in SysML v2.

The SysML v2 use cases in **Figure 19** correspond to the SysML v1 use cases in **Figure 18**. The use case definitions are depicted using the standard definition symbol with compartments for its objectives, subject, and actors. The use case specializations are depicted with the specialization symbol.

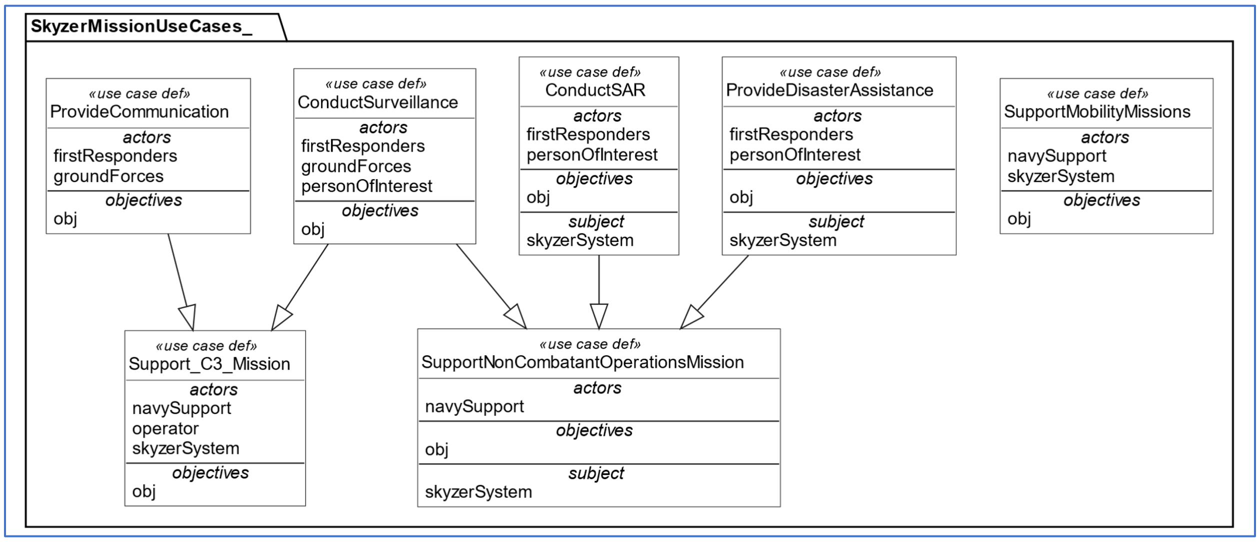


Figure 19. Skyzer Mission Use Cases in SysML v2

The Use Case Number can be mapped to a short name in the same way that was shown for the requirement id. The SysML v1 stereotypes and their properties can be mapped to corresponding elements using the SysML v2 language extension mechanism that is described in the Transform Stereotypes section below.

4.1.7 Transform Activities

The next step in the transformation was to transform the activities which are generally shown in SysML v1 activity diagrams. The activities in the Skyzer Mission Model are contained in the Mission Behavior package and some additional activities are contained in the Use Cases package. A partial view of the SysML v1 activity diagram called 'Non-Combatant Operations - Scenario 1' is shown in **Figure 20**. This activity is intended to realize the SysML v1 use case in **Figure 18** called Support Noncombatant Operations Mission.

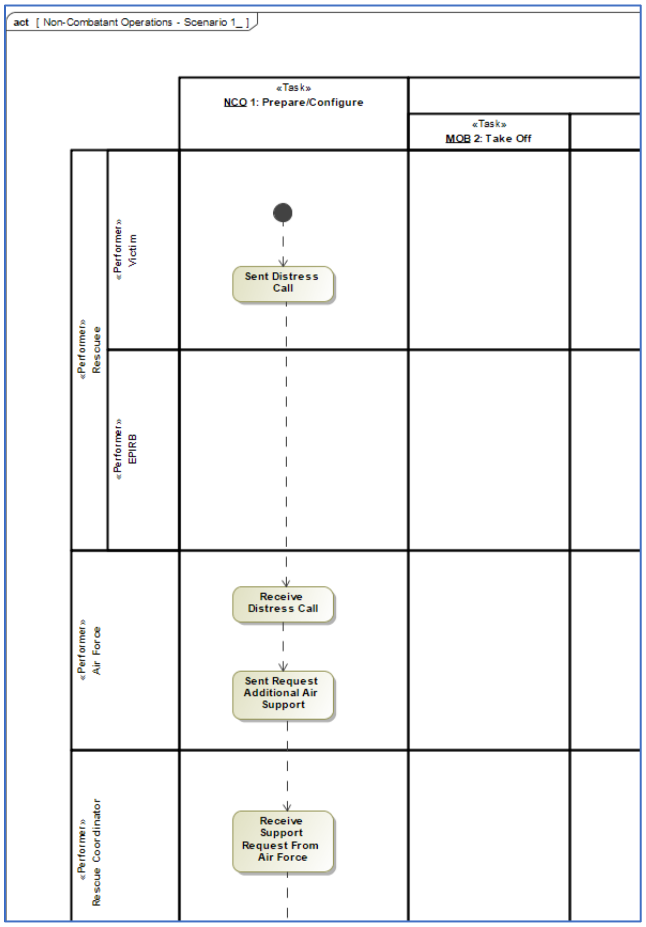


Figure 20. Activity Diagram for Non-Combatant Operations Scenario in SysML v1

This activity contains actions with control flows and one object flow (not shown). In parts of the activity diagram, there are multiple control flows that connect to a single action (not shown). This is often modeled by connecting multiple flows to a join node and then connecting the outgoing edge of the join node to an action, but there were no join nodes or other control nodes in this model.

This activity diagram contains two sets of swim lanes. The horizontal swim lanes are intended to correspond to various elements in the Skyzer Mission domain that the activities are allocated to. However, the swim lanes are not related to the blocks in this model. The vertical swim lanes correspond to a partitioning of actions into other activities, but again there was no relation between the swim lane and any other activity.

The SysML v1 activities transform to SysML v2 action definitions and the SysML v1 actions transform to SysML v2 actions. The initial transformation only included the activity and its decomposition into actions. A partial view of this decomposition is shown in **Figure 21**. The swim lanes were not included in the initial transformation and deferred to the post-processing where the swim lanes can be better integrated into the model. This swim lanes were removed as part of the pre-processing.

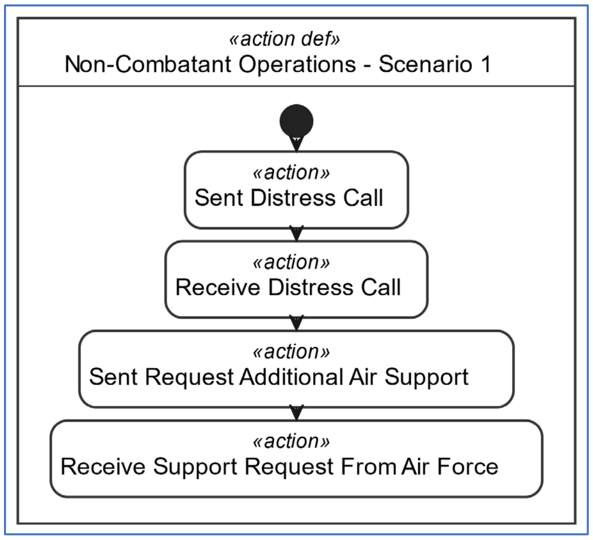

«action def»
Non-Combatant Operations - Scenario 1
«action»
Sent Distress Call
«action»
Receive Distress Call
«action»
Sent Request Additional Air Support
«action»
Receive Support Request From Air Force

Figure 21. Action Flow for Non-Combatant Operations Scenario in SysML v2 (partial)

4.1.8 Transform Interactions (e.g., sequence diagrams)

The next step in the transformation was to transform the interactions which are generally shown in SysML v1 sequence diagrams. The use case called ‘Activate and Launch UAV’ in **Figure 22** in the Mission Use Cases package contains an activity diagram in **Figure 23** called ‘Activate and Launch UAV’ and a sequence diagram in **Figure 24** with the same name. The sequence diagram is intended to be a further refinement of the activity diagram.

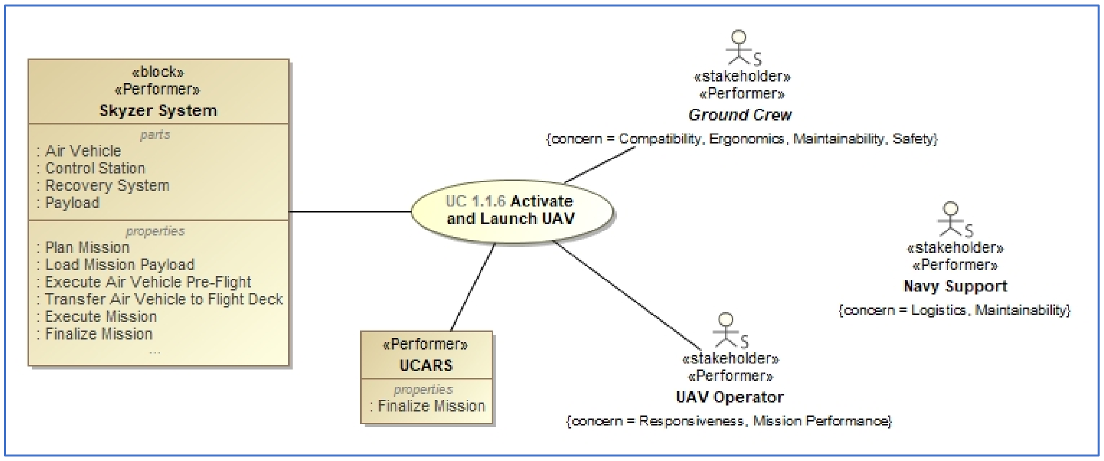


Figure 22. Activate and Launch UAV Use Case Diagram Contains the Activate and Launch Sequence Diagram in Figure 19b (partial)

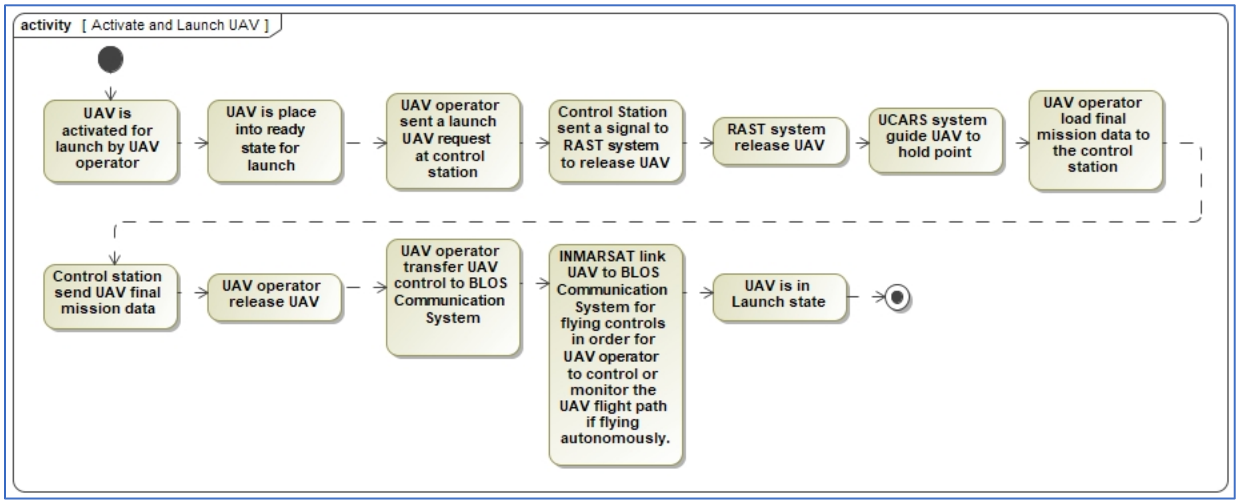

activity [Activate and Launch UAV]
UAV is activated for launch by UAV operator
UAV is place into ready state for launch
UAV operator sent a launch UAV request at control station
Control Station sent a signal to RAST system to release UAV
RAST system release UAV
UCARS system guide UAV to hold point
UAV operator load final mission data to the control station
Control station send UAV final mission data
UAV operator release UAV
UAV operator transfer UAV control to BLOS Communication System
INMARSAT link UAV to BLOS Communication System for flying controls in order for UAV operator to control or monitor the UAV flight path if flying autonomously.
UAV is in Launch state

Figure 23. Activate and Launch UAV Activity Diagram in SysML v1 (partial)

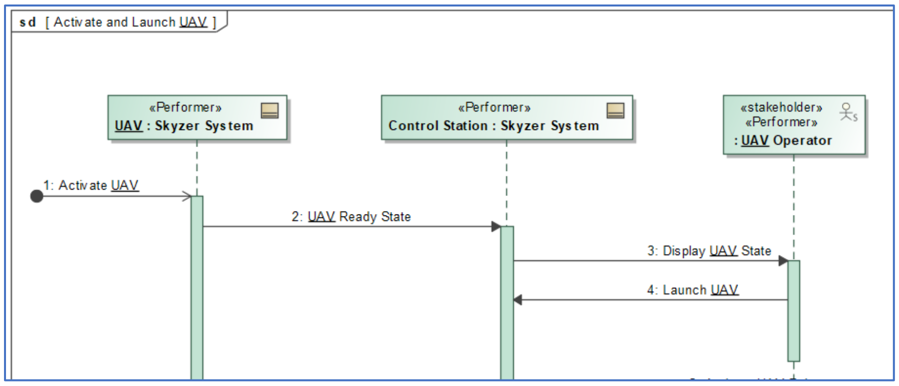


Figure 24. Activate and Launch UAV Sequence Diagram in SysML v1 (partial)

An interaction in SysML v1 transforms to an occurrence in SysML v2. The SysML v1 parts and their lifelines transform to parts in SysML v2. The SysML v1 messages transform to SysML v2 messages.

Figure 25 depicts the corresponding model fragment in SysML v2 for the SysML v1 sequence diagram in **Figure 24**. The rendering in Plant UML uses the compartment notation to depict the parts that are interacting, the messages (e.g., flows) between them, and the message sequence. Commercial tools are expected to support the more standard visualization shown in **Figure 24**.

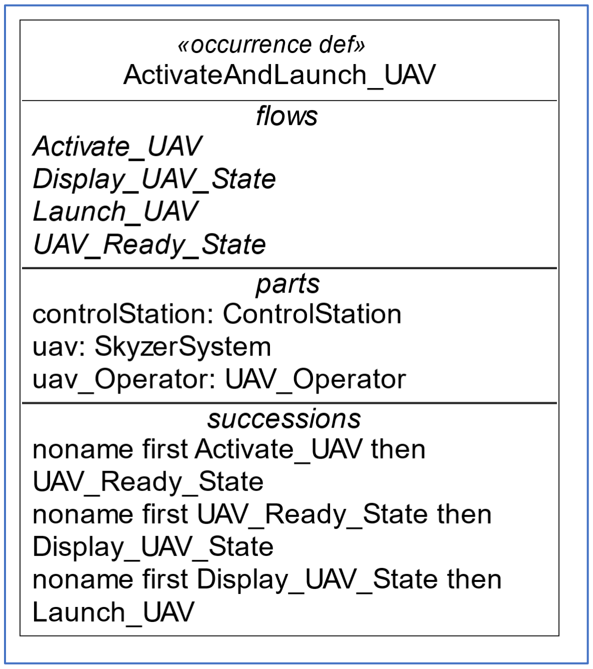

«occurrence def»
ActivateAndLaunch_UAV
flows
Activate_UAV
Display_UAV_State
Launch_UAV
UAV_Ready_State
parts
controlStation: ControlStation
uav: SkyzerSystem
uav_Operator: UAV_Operator
successions
noname first Activate_UAV then UAV_Ready_State
noname first UAV_Ready_State then Display_UAV_State
noname first Display_UAV_State then Launch_UAV

Figure 25. Activate and Launch UAV Sequence in SysML v2 (partial)

Note: This will be represented by a more conventional sequence diagram in commercial tools

4.1.9 Transform State-Based Behavior

The next step in the transformation was to transform the state-based behaviors which are generally shown in SysML v1 state machine diagrams. States in SysML v1 transform to states in SysML v2 and transitions in SysML v1 transform to transitions in SysML v2. The entry, exit, and do behaviors and transition effects in SysML v1 generally transform into corresponding actions in SysML v2. There are no state-based behaviors in this model.

4.1.10 Transform Parametrics

The next step in the transformation was to transform the parametric constraints which are generally shown in SysML v1 parametric diagrams. The constraint blocks and constraint properties in SysML v1 transform to constraint definitions and constraint usages in SysML v2. The constraint parameters in SysML v1 transform to input parameters of the constraint in SysML v2. Binding connectors in SysML v1 transform to binding connections in SysML v2. There are no parametrics in this model.

4.1.11 Transform Requirements Relationships

The next step in the transformation is to transform the requirements relationships including satisfy, verify, derive, refine, trace, and copy. There were several trace relationships used between requirements in the SysML v1 Skyzer model. The trace relationship was mapped to a corresponding dependency relationship in the transformed SysML v2 model. **Figure 26**, is an example of the dependency relationship using the textual notation.

```
dependency from operationalRequirements::imagingCapability to missionRequirments::imagingCapability;
```

Figure 26. Example of the Dependency Relationship Using the Textual Notation

Figure 27 depicts the SysML v1 measures of effectiveness previously shown in **Figure 12** and the requirements they are asserted to satisfy. For example, the cruise speed MOE is asserted to satisfy the Cruise Speed requirement.

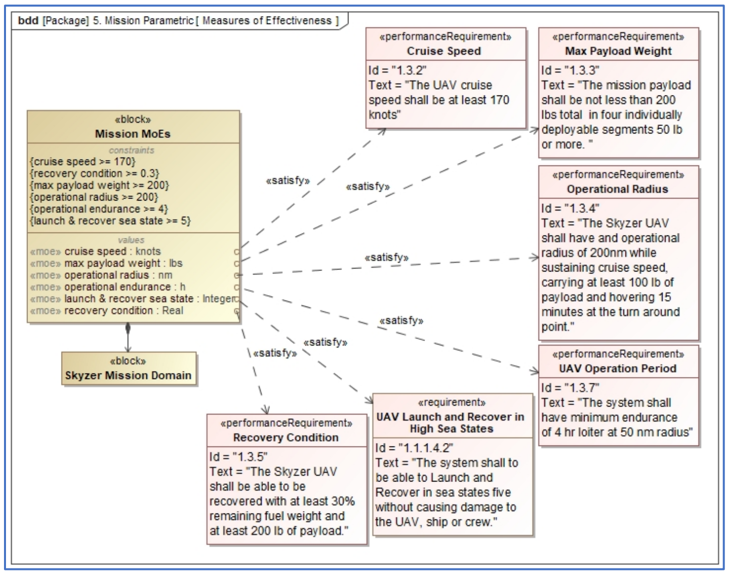

bdd [Package] 5. Mission Parametric [Measures of Effectiveness]
«block»
Mission MoEs
constraints
{cruise speed >= 170}
{recovery condition >= 0.3}
{max payload weight >= 200}
{operational radius >= 200}
{operational endurance >= 4}
{launch & recover sea state >= 5}
values
«moe» cruise speed : knots
«moe» max payload weight : lbs
«moe» operational radius : nm
«moe» operational endurance : h
«moe» launch & recover sea state : Integer
«moe» recovery condition : Real
«block»
Skyzer Mission Domain
«satisfy»
«satisfy»
«satisfy»
«satisfy»
«satisfy»
«satisfy»
«performanceRequirement»
Cruise Speed
Id = "1.3.2"
Text = "The UAV cruise speed shall be at least 170 knots"
«performanceRequirement»
Max Payload Weight
Id = "1.3.3"
Text = "The mission payload shall be not less than 200 lbs total in four individually deployable segments 50 lb or more. "
«performanceRequirement»
Operational Radius
Id = "1.3.4"
Text = "The Skyzer UAV shall have and operational radius of 200nm while sustaining cruise speed, carrying at least 100 lb of payload and hovering 15 minutes at the turn around point."
«performanceRequirement»
UAV Operation Period
Id = "1.3.7"
Text = "The system shall have minimum endurance of 4 hr loiter at 50 nm radius"
«requirement»
UAV Launch and Recover in High Sea States
Id = "1.1.1.4.2"
Text = "The system shall to be able to Launch and Recover in sea states five without causing damage to the UAV, ship or crew."
«performanceRequirement»
Recovery Condition
Id = "1.3.5"
Text = "The Skyzer UAV shall be able to be recovered with at least 30% remaining fuel weight and at least 200 lb of payload."

Figure 27. Requirements Satisfied by the Measures of Effectiveness in SysML v1

The transformation specification transforms a satisfy relationship in SysML v1 to a satisfy requirement usage in SysML v2. This satisfy requirement usage has significantly more semantics than the SysML v1 satisfy relationship and includes the ability to evaluate whether the requirement is satisfied. For this transformation, a SysML v1 satisfy relationship was transformed to a requirement allocation relationship in SysML v2. The allocation relationship is used more like a SysML v1 satisfy relationship.

Figure 28 shows the SysML v2 measures of effectiveness and the requirements that are allocated to them. The Plant UML visualization renders the MOEs, the requirements, and their allocation relationships in the compartment notation.

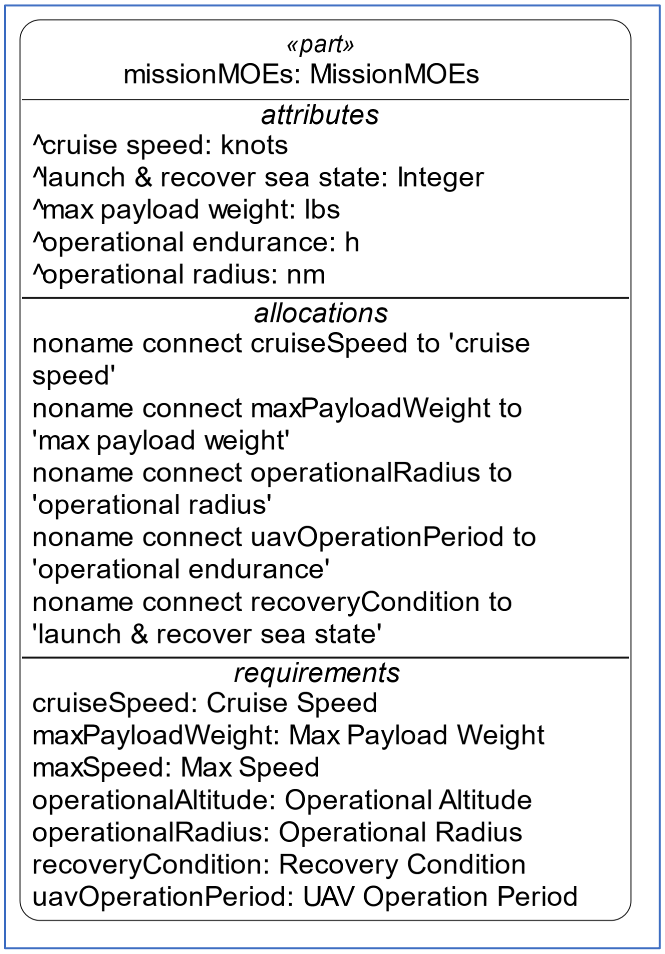

«part»
missionMOEs: MissionMOEs
attributes
^cruise speed: knots
^launch & recover sea state: Integer
^max payload weight: lbs
^operational endurance: h
^operational radius: nm
allocations
noname connect cruiseSpeed to 'cruise speed'
noname connect maxPayloadWeight to 'max payload weight'
noname connect operationalRadius to 'operational radius'
noname connect uavOperationPeriod to 'operational endurance'
noname connect recoveryCondition to 'launch & recover sea state'
requirements
cruiseSpeed: Cruise Speed
maxPayloadWeight: Max Payload Weight
maxSpeed: Max Speed
operationalAltitude: Operational Altitude
operationalRadius: Operational Radius
recoveryCondition: Recovery Condition
uavOperationPeriod: UAV Operation Period

Figure 28. Requirements that Are Allocated to the Measures of Effectiveness in SysML v2

The verify, derive, and refine relationships in SysML v1 map to corresponding verify, derive, and refine relationships in SysML v2. A trace requirement can be mapped to a dependency. A copy relationship is no longer required since SysML v2 enables reuse of a requirement with requirements usages.

4.1.12 Transform Other Elements

Most of the primary elements in SysML v2 would have been transformed through the previous steps. The next step in the transformation was to transform other elements that were not addressed by the previous transformation steps.

One example is the transformation of the SysML v1 Mission Data Model in **Figure 29** that is contained in package within the Support Elements package. A data element is represented in the SysML v1 model as a UML class. This requires pre-processing to convert each data element to a SysML v1 block. The block is then converted to a SysML v2 item definition. The data elements in the SysML v1 model are related by stereotyped dependencies in this example, although they a more rigorous conversion may relate the date elements using connection definitions similar to associations in SysML v1.

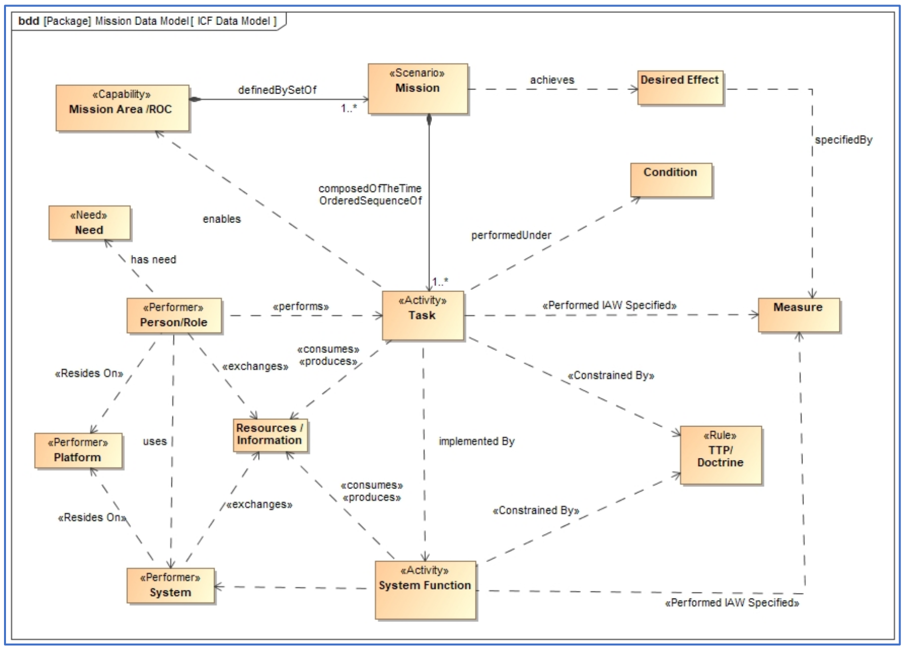


Figure 29. Mission Data Model in SysML v1

The corresponding Mission Data Model in SysML v2 is shown in **Figure 30**. The stereotyped dependencies in the SysML v1 model are mapped to metadata that is displayed as a key word. Some, but not all the stereotypes were mapped, such as *Mission achieves Desired Effect.*

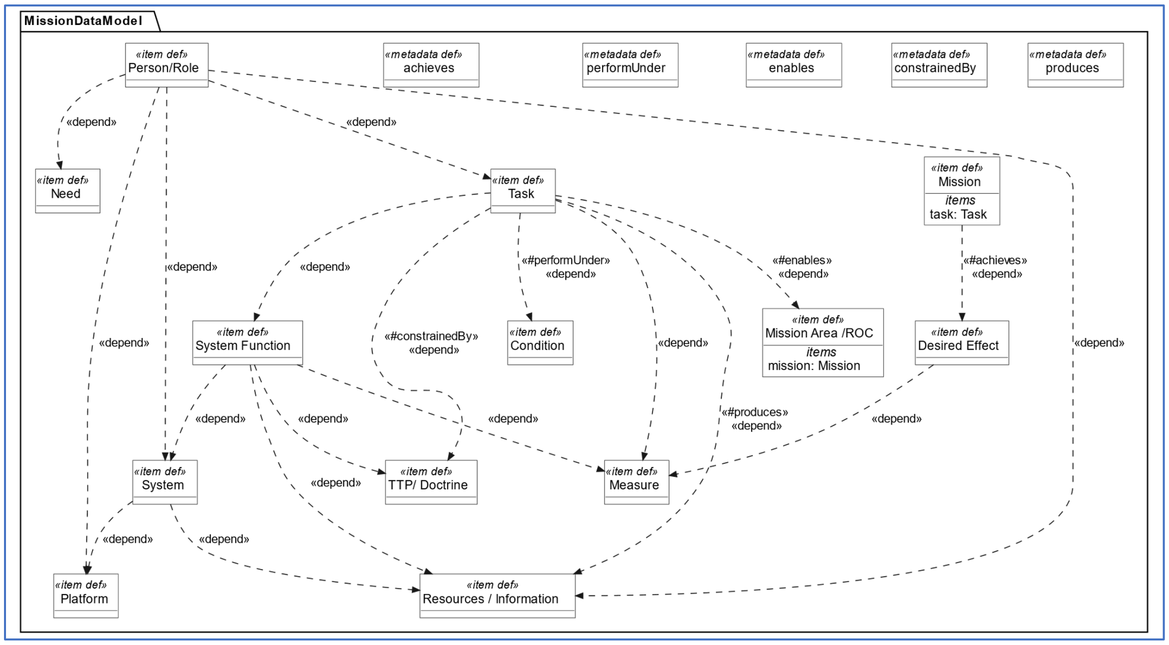


Figure 30. Mission Data Model in SysML v2

4.1.13 Transform Stereotypes

The SysML v1 Skyzer Mission Model contains several stereotypes. Some of the stereotypes include stereotype properties. For example, the SysML v1 model includes an extension of a requirement called a JCIDS requirement that includes a property to identify the type of requirement. The type of requirement is an enumeration that includes the set of values KPP, KSA, OSA, and APA. The stereotype definition is shown in **Figure 31** along with an example of the stereotype applied to *RequirementA*.

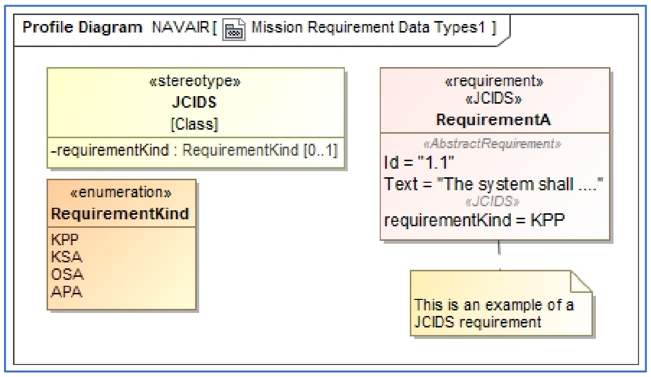
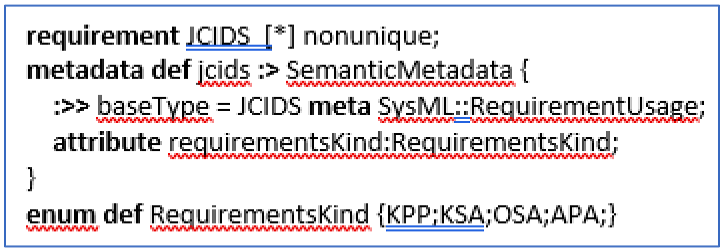

Profile Diagram NAVAIR [Mission Requirement Data Types1]
«stereotype»
JCIDS
[Class]
-requirementKind : RequirementKind [0..1]
«enumeration»
RequirementKind
KPP
KSA
OSA
APA
«requirement»
«JCIDS»
RequirementA
«AbstractRequirement»
Id = "1.1"
Text = "The system shall ...."
«JCIDS»
requirementKind = KPP
This is an example of a JCIDS requirement

Figure 31. A SysML v1 Example of an applied stereotype to Requirement A

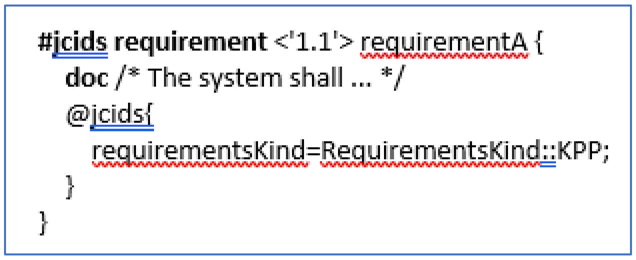
SysML v1 stereotypes map to corresponding concepts that extend SysML v2 concepts. The JCIDS requirement extension in SysML v2 is shown below in **Figure 32**, using its metadata extension mechanism. JCIDS is defined as a requirement. The metadata def enables *jcids* to be used as a key word. The *requirementsKind* property is defined by an enumeration definition that identifies a valid set of discrete values.



```
requirement JCIDS [*] nonunique;
metadata def jcids :> SemanticMetadata {
   :>> baseType = JCIDS meta SysML::RequirementUsage;
   attribute requirementsKind:RequirementsKind;
}
enum def RequirementsKind {KPP;KSA;OSA;APA;}
```

Figure 32. JCIDS Requirement Extension in SysML v2

The example below in **Figure 33**, applies the *jcids* key word to the requirement called *requirementA* with an id (e.g., short name) of ‘1.1’. The abbreviated text statement is included following the key word **doc**. The *requirementsKind* is a KPP. The graphical notation is similar to the SysML v1 notation in **Figure 31**.



```
#jcids requirement <'1.1'> requirementA {
   doc /* The system shall ... */
   @jcids{
      requirementsKind=RequirementsKind::KPP;
   }
}
```

Figure 33. Example Applies the *jcids* Key Word to the Requirement Called RequirementA with an ID (e.g., short name) of ‘1.1’

4.1.14 Transform Customizations

The SysML v1 model includes tool-specific customizations such as a glossary, acronym list, a legend, and some custom images. In general, customizations will require pre-processing and special mapping rules. For example, the list of acronyms is defined as ‘terms’ as part of the tool customization. Each term could be mapped to an alias in SysML v2. An example is the term CCC that is an acronym for Command, Control, and Communications in the SysML v1 model. The term can be mapped to an alias of Command, Control, and Communications in the SysML v2 model as follows: **alias** CCC for 'Command, Control and Communication';

There are other elements such as views and viewpoints, which were not in the SysML v1 model and therefore were not addressed by the transformation.

4.2 Post-Process the SysML v2 Skyzer Model

This step involved post-processing the transformed SysML v2 model to take advantage of some of the SysML v2 modeling capabilities. The SysML v2 model was significantly reorganized and the model was refactored to align with the usage focused modelling paradigm.

The usage focused modeling paradigm leverages the SysML v2 definition and usage pattern that supports decomposition and specialization of parts, actions, requirements, and many other kinds of SysML v2 elements. A decomposition with usage focused modeling results in a hierarchy of parts, actions, requirements, etc. This contrasts with the block decomposition in SysML v1 which decomposes blocks into parts that are typed by blocks, and those blocks are further decomposed into parts that are typed by blocks. The more direct part decomposition in SysML v2 results in a straightforward parts tree. In the usage focused paradigm, the parts can be defined by part definitions, but each part definition represents a black box that does not contain parts of its own. This enables multiple parts to be defined by the same black box part definition but have their own part decomposition. This approach facilitates reuse of the black box specifications and can be applied at each level of design.

Similarly, a SysML v1 activity decomposes into call behavior actions that call activities that further decompose into call behavior actions. This again contrasts with a more direct-action decomposition in SysML v2 resulting in an action tree. The actions can be defined by action definitions which specify their inputs and outputs. This same usage focused decomposition pattern applies to virtually all SysML v2 concepts.

The model organization for a usage focused paradigm separates the packages that contain definition elements from the packages that use the definition elements. The packages that use the definition elements contain the usage hierarchies such as a parts tree and action tree, and the cross connections such as between parts and actions.

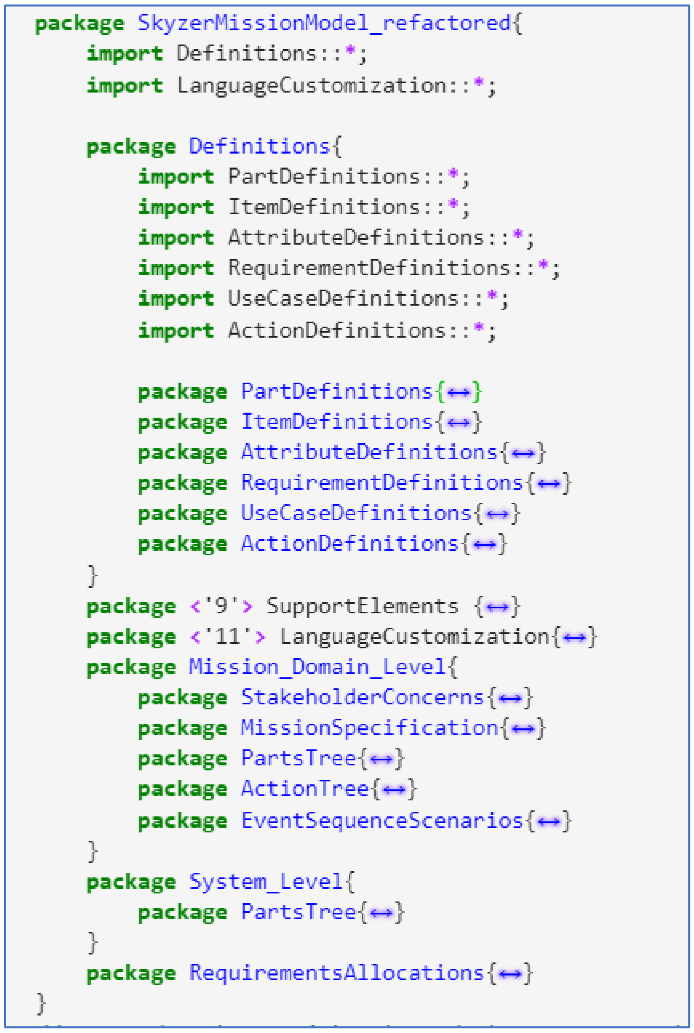
The post-processing steps were performed incrementally as follows:

- Reorganize the SysML v2 model packages
- Refactor parts hierarchy
- Refactor parts interconnection
- Capture action definitions in action definitions package
- Refactor action hierarchy
- Integrate behavior
- Refactor the requirements
- Refactor requirements traceability
- Additional refactoring

4.2.1 Reorganize The SysML v2 Model Packages

The first post-processing step was to establish a new package structure to begin re-organizing the SysML v2 model. The new package structure takes advantage of the usage focused modeling paradigm which was briefly summarized in section 4.2.

As shown in **Figure 34**, this package structure separates the definition elements from the usage elements. The Definitions package contains nested packages for *PartDefinitions*, *ItemDefinitions*, *AttributeDefinitions*, *RequirementDefinitions*, *UseCaseDefinitions*, *ActionDefintions,* and other specialized definition packages which are not shown. The usage of the definition elements is contained in the *Mission_Domain_Level*, *System_Level*, and *ReqirementsAllocations* packages.



```
package SkyzerMissionModel_refactored{
    import Definitions::*;
    import LanguageCustomization::*;

    package Definitions{
        import PartDefinitions::*;
        import ItemDefinitions::*;
        import AttributeDefinitions::*;
        import RequirementDefinitions::*;
        import UseCaseDefinitions::*;
        import ActionDefinitions::*;

        package PartDefinitions{↔}
        package ItemDefinitions{↔}
        package AttributeDefinitions{↔}
        package RequirementDefinitions{↔}
        package UseCaseDefinitions{↔}
        package ActionDefinitions{↔}
    }
    package <'9'> SupportElements {↔}
    package <'11'> LanguageCustomization{↔}
    package Mission_Domain_Level{
        package StakeholderConcerns{↔}
        package MissionSpecification{↔}
        package PartsTree{↔}
        package ActionTree{↔}
        package EventSequenceScenarios{↔}
    }
    package System_Level{
        package PartsTree{↔}
    }
    package RequirementsAllocations{↔}
}
```

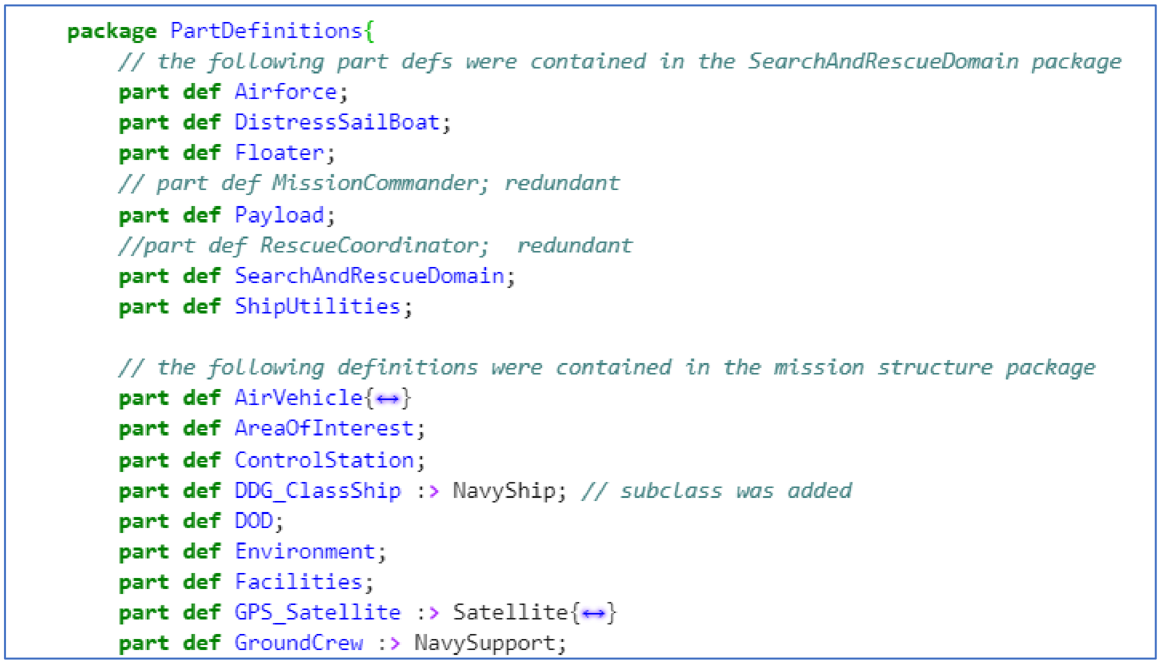
The package structure is reorganized to separate the definition elements from the usage elements and take advantage of the usage-focused modeling paradigm

Figure 34. Package Structure

The *Mission_Domain_Level* package contains nested packages for *StakeholderConcerns*, *MissionSpecification*, *PartsTree*, *ActionTree*, and *EventSequenceeScenarios*, which each contain usage elements defined by definition elements that are contained in the Definitions package.

The *SupportElements* package contains the Glossary, References, and other supporting information that was included in the original SysML v1 model. The *LanguageCustomization* package contains the key word extensions. The package number from the original SysML v1 model is captured as a short name in brackets for both the *SupportElements* and *LanguageCustomization* packages. This numbering can be applied consistently to all packages if desired but was included here to highlight a use of this language feature.

Capture part definitions into PartDefinitions package. The next post-processing step was to capture the part definitions that were contained in multiple packages in the transformed model in the *PartDefinitions* package. The parts are deleted from each part definition so that this package contains a flat list of part definitions with no hierarchy as shown in the partial list of part definitions in **Figure 35**. The hierarchy is reconstituted in the next step as a parts hierarchy. As part of this reorganization, several redundant part definitions were identified, and the redundant elements were carefully deleted to ensure that the appropriate relationships were reflected in the model.



```
package PartDefinitions{
    // the following part defs were contained in the SearchAndRescueDomain package
    part def Airforce;
    part def DistressSailBoat;
    part def Floater;
    // part def MissionCommander; redundant
    part def Payload;
    //part def RescueCoordinator;  redundant
    part def SearchAndRescueDomain;
    part def ShipUtilities;

    // the following definitions were contained in the mission structure package
    part def AirVehicle{↔}
    part def AreaOfInterest;
    part def ControlStation;
    part def DDG_ClassShip :> NavyShip; // subclass was added
    part def DOD;
    part def Environment;
    part def Facilities;
    part def GPS_Satellite :> Satellite{↔}
    part def GroundCrew :> NavySupport;
```

The Part Definitions package contains a flat list of part definitions that were contained in multiple other packages in the transformed model.

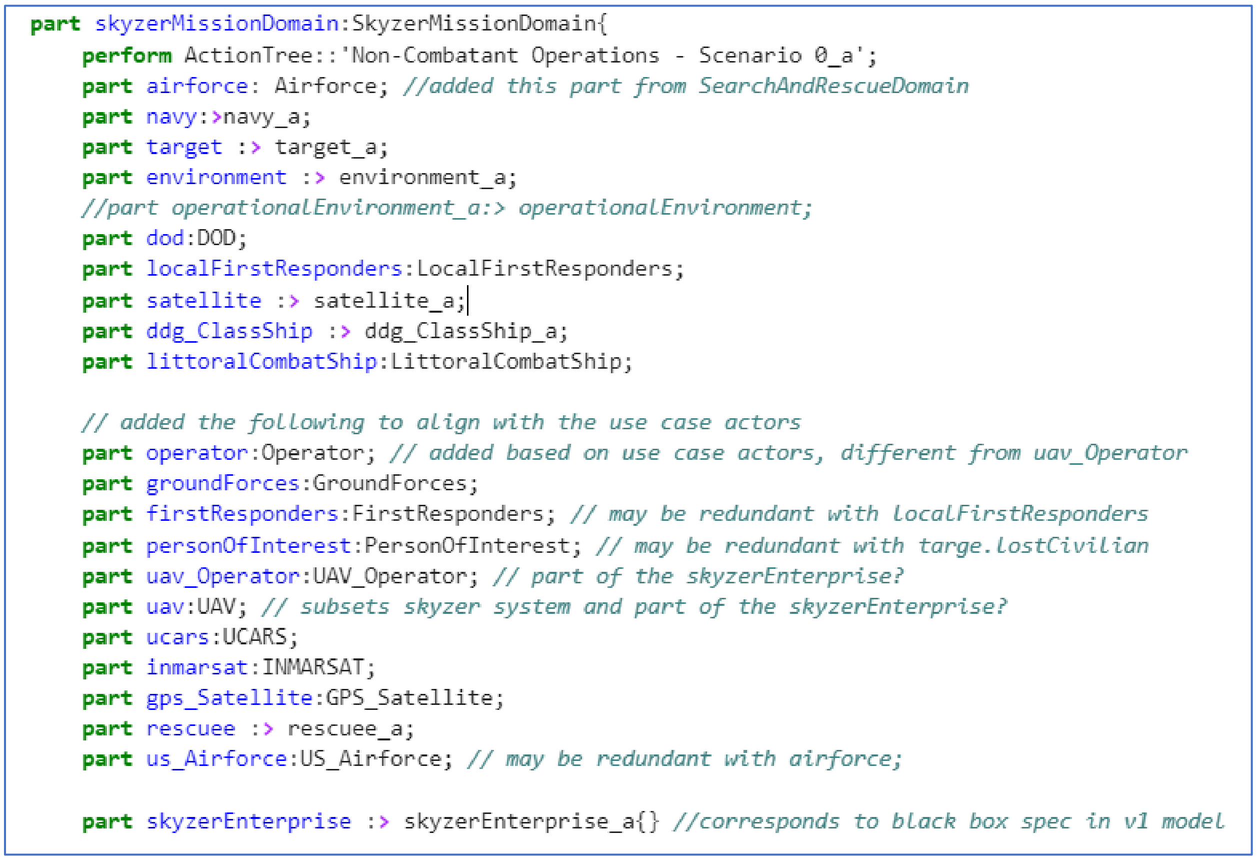
Figure 35. Part Definitions Package

4.2.2 Refactor Parts Hierarchy

The next post-processing step was to reconstitute the system hierarchy at the mission and system level by creating a parts tree. The parts tree is intended to correspond to the original block decomposition structure in SysML v1.

However, multiple inconsistencies from the original SysML v1 block hierarchy were identified and reconciled. For example, the block hierarchy for the OV-1 in **Figure 5**, the Skyzer Mission Domain in **Figure 6**, the Skyzer Black Box Interfaces in **Figure 10**, and the block hierarchy associated with the swim lanes in the activity diagram in **Figure 20** were not the same.

The top of the refactored part hierarchy is the part called *SkyzerMissionDomain* which is shown in **Figure 36**. The *skyzerEnterprise* subsets *skyzerEnterprise_a* which is further decomposed to include the skyzerSystem as shown in **Figure 37**. The skyzerSystem subsets the *skyzerSystem_a* parts tree which is shown in **Figure 38**. The parts hierarchy is reconciled as a single consistent parts tree where the parts are defined by part definitions. The perform action in **Figure 36** was added later in the process.

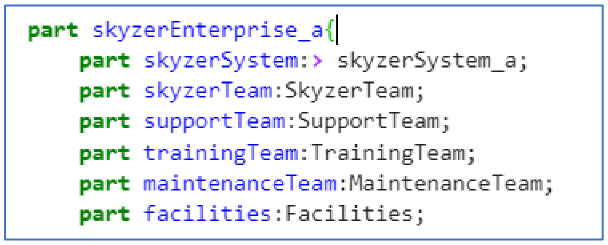

```
part skyzerMissionDomain:SkyzerMissionDomain{
    perform ActionTree::'Non-Combatant Operations - Scenario 0_a';
    part airforce: Airforce; //added this part from SearchAndRescueDomain
    part navy:>navy_a;
    part target :> target_a;
    part environment :> environment_a;
    //part operationalEnvironment_a:> operationalEnvironment;
    part dod:DOD;
    part localFirstResponders:LocalFirstResponders;
    part satellite :> satellite_a;
    part ddg_ClassShip :> ddg_ClassShip_a;
    part littoralCombatShip:LittoralCombatShip;

    // added the following to align with the use case actors
    part operator:Operator; // added based on use case actors, different from uav_Operator
    part groundForces:GroundForces;
    part firstResponders:FirstResponders; // may be redundant with localFirstResponders
    part personOfInterest:PersonOfInterest; // may be redundant with targe.lostCivilian
    part uav_Operator:UAV_Operator; // part of the skyzerEnterprise?
    part uav:UAV; // subsets skyzer system and part of the skyzerEnterprise?
    part ucars:UCARS;
    part inmarsat:INMARSAT;
    part gps_Satellite:GPS_Satellite;
    part rescuee :> rescuee_a;
    part us_Airforce:US_Airforce; // may be redundant with airforce;

    part skyzerEnterprise :> skyzerEnterprise_a{} //corresponds to black box spec in v1 model
```

The top of the refactored part hierarchy is the part called SkyzerMissionDomain

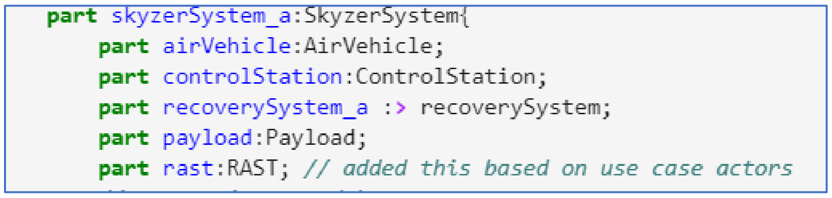
Figure 36. Skyzer Mission Domain Parts Tree



```
part skyzerEnterprise_a{
    part skyzerSystem:> skyzerSystem_a;
    part skyzerTeam:SkyzerTeam;
    part supportTeam:SupportTeam;
    part trainingTeam:TrainingTeam;
    part maintenanceTeam:MaintenanceTeam;
    part facilities:Facilities;
```

The skyzerEnterprise_a parts tree which includes the skzyerSystem part

Figure 37. Skyzer Enterprise Parts Tree



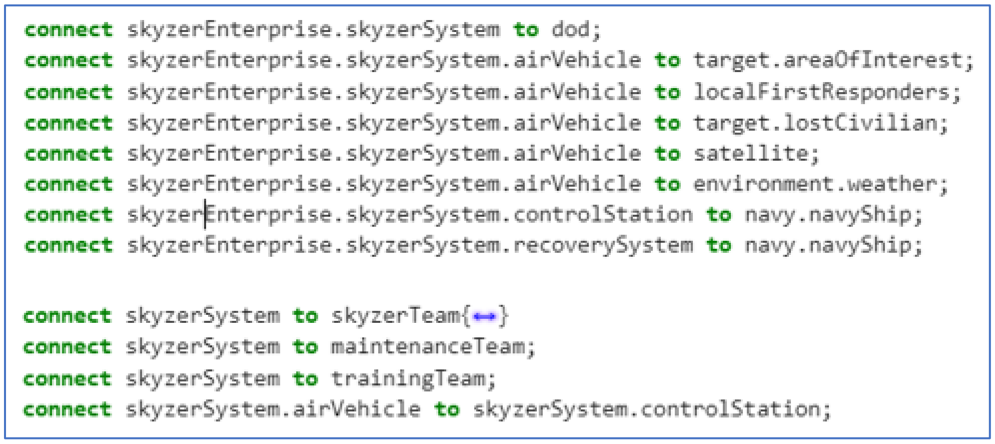
```
part skyzerSystem_a:SkyzerSystem{
    part airVehicle:AirVehicle;
    part controlStation:ControlStation;
    part recoverySystem_a :> recoverySystem;
    part payload:Payload;
    part rast:RAST; // added this based on use case actors
```

Figure 38. Skyzer System Parts Tree

4.2.3 Refactor Parts Interconnection

The next post-processing step was to reconstitute the system interconnection at the mission and system level. The parts interconnection view is intended to correspond to a refinement of the original SysML internal block diagrams (ibd) at the mission and system level.

The SysML v1 Skyzer model did not include a mission level ibd. It did include the Skyzer OV-1 in **Figure 5** which implied the interconnection but used dependencies between parts instead of connectors. The corresponding SysML v2 view was shown in **Figure 7**. The SysML v2 connections were added as shown in **Figure 39** to ensure a consistent structural representation that connects the system and its parts to other external parts that are part of the mission and enterprise.



```
connect skyzerEnterprise.skyzerSystem to dod;
connect skyzerEnterprise.skyzerSystem.airVehicle to target.areaOfInterest;
connect skyzerEnterprise.skyzerSystem.airVehicle to localFirstResponders;
connect skyzerEnterprise.skyzerSystem.airVehicle to target.lostCivilian;
connect skyzerEnterprise.skyzerSystem.airVehicle to satellite;
connect skyzerEnterprise.skyzerSystem.airVehicle to environment.weather;
connect skyzerEnterprise.skyzerSystem.controlStation to navy.navyShip;
connect skyzerEnterprise.skyzerSystem.recoverySystem to navy.navyShip;

connect skyzerSystem to skyzerTeam{↔}
connect skyzerSystem to maintenanceTeam;
connect skyzerSystem to trainingTeam;
connect skyzerSystem.airVehicle to skyzerSystem.controlStation;
```

The connections between the system and its parts and the parts of the mission and enterprise

Figure 39. Connections

4.2.4 Capture Action Definitions in the ActionDefinitions Package

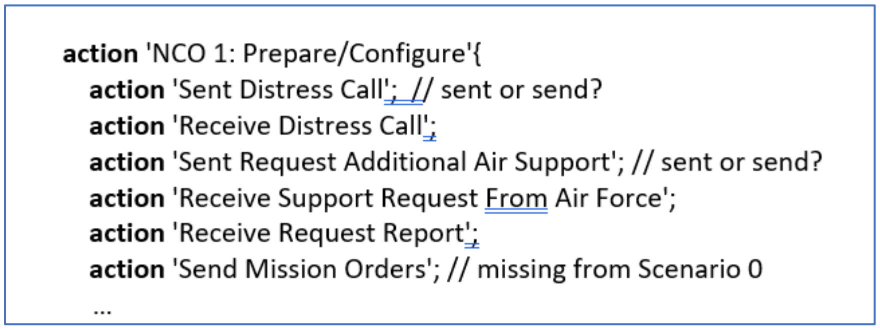
The next post-processing step was to capture the action definitions from the transformed model in the *ActionDefinitions* package. The activities in the SysML v1 model are contained in the Mission Behavior and Use Case package. The activities contain actions, control nodes, control flows, object flows, and swim lanes.

A partial view of one of the activities called ‘Non-Combatant Operations - Scenario 1’ is shown in **Figure 20**. This activity and associated action hierarchy was significantly refactored as described in the next section. To accommodate this refactoring, an action definition was created to correspond to each nested action contained in the activity to enable the same action definitions to be used in more than one action flow view. As was done with the *PartDefinitions* package, the *ActionDefinitions* package contains a flat list of action definitions without any hierarchy. The hierarchy is reconstituted in a follow-on section.

4.2.5 Refactor Action Hierarchy

The action hierarchy from the transformed model in **Figure 21** was refactored through a series of steps. First, there were two action definitions in the Mission Behavior package called ‘Non-Combatant Operations - Scenario 1’ and ‘Non-Combatant Operations - Scenario 2’. Each of the action definitions contained multiple actions. Scenario 1 contained 56 actions and Scenario 2 contained 55 actions. After further analysis, it was noted that both Scenarios were the same except that Scenario 1 contained an additional action called 'Release UAV to Second Location'. An action called Scenario 0 was created to capture the common actions, action hierarchy, control nodes, control flow, and object flows enabling Scenario 1 and Scenario 2 to subset Scenario 0 and inherit its common features and modify as needed.

The activities for Scenario 1 and Scenario 2 from the original SysML v1 model include two sets of swim lanes. The vertical swim lanes represent a further partitioning of the actions into other actions. For example, the first vertical swim lane in the activity diagram in **Figure 20** is called ‘NCO 1: Prepare/Configure’. In the refactored model, these actions are part of the action hierarchy. For example, the nested actions contained in the ‘NCO 1: Prepare/Configure’ swim lane are nested in the action called 'NCO 1: Prepare/Configure' as shown in **Figure 40**.

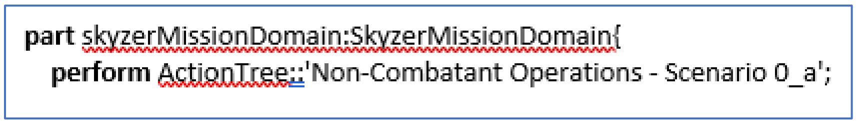


```
action 'NCO 1: Prepare/Configure'{
  action 'Sent Distress Call'; // sent or send?
  action 'Receive Distress Call';
  action 'Sent Request Additional Air Support'; // sent or send?
  action 'Receive Support Request From Air Force';
  action 'Receive Request Report';
  action 'Send Mission Orders'; // missing from Scenario 0
  ...
```

The action ‘NCO 1: Prepare/Configure’ was represented as a swim lane in the SysML v1 model but is represented as an action with nested actions in SysML v2 that is part of the action hierarchy.

Figure 40. Action ‘NCO 1: Prepare/Configure’

The horizontal swim lanes are intended to represent blocks that the activities are allocated to. However, the allocation was not explicitly included in the SysML v1 model. The implied allocation was refactored to represent an action that is performed by a part. This is illustrated in **Figure 36** and shown below in **Figure 41**, where the part called *skyzerMissionDomain* contains a perform action that refers to the action called ‘Non-Combatant Operations - Scenario 0_a' that is contained in the *ActionTree* package.



```
part skyzerMissionDomain:SkyzerMissionDomain{
  perform ActionTree::'Non-Combatant Operations - Scenario 0_a';
```

The part called *skyzerMissionDomain* contains a perform action.

Figure 41. Perform Action

Each allocated action from each swim lane from the original SysML v1 model can be transformed into a perform action of a part that corresponds to the swim lane. This was not done as part of the transformation process since the part hierarchy was significantly altered during the post-processing, and this would have created additional work to reassign the perform actions to the parts in the refactored part hierarchy. Instead, this was done as part of the next step to integrate behavior.

4.2.6 Integrate Behavior

The SysML v1 Skyzer model captures behavior in terms of use cases, activity diagrams, and sequence diagrams. The transformed model captures each of these behaviors. However, there are many opportunities to integrate the behavior in SysML v2 including the use cases, action flow, message/event sequences, and state-based behavior.

One example of an opportunity to integrate behavior in the Skyzer Model is the behavior associated with activating and launching the UAV. There is a use case called 'Activate and Launch UAV' which contains an activity diagram and a sequence diagram with the same name. The use case diagram and a portion of the activity diagram and a portion of the sequence diagram are shown in **Figure 22**, **Figure 23**, and **Figure 24** respectively.

Another activity diagram in the Mission Behavior package called 'Non-Combatant Operations - Scenario 1' also contains actions to 'Activate UAV', 'Launch UAV', and several other actions that are part of the Activate and Launch sequence. A portion of this activity was shown in **Figure 20** but the actions to activate and launch the UAV were not shown. The actions that support activate and launch are nested within the vertical swim lane called 'MOB 2: Take Off', which in turn is nested within the vertical swim lane called 'NCO 6: Respond to Emergencies'. In addition, these actions have an implied allocation to various blocks based on the horizontal swim lanes that they are contained in.

The use case, activity diagrams, and sequence diagram related to activate and launch in the SysML v1 model contain separate elements that are not explicitly related to one another. Ensuring consistency among elements from different diagram kinds (e.g., activity, sequence, state, use case) is often difficult to do within the SysML v1 model, but there are opportunities to integrate this behavior in SysML v2.

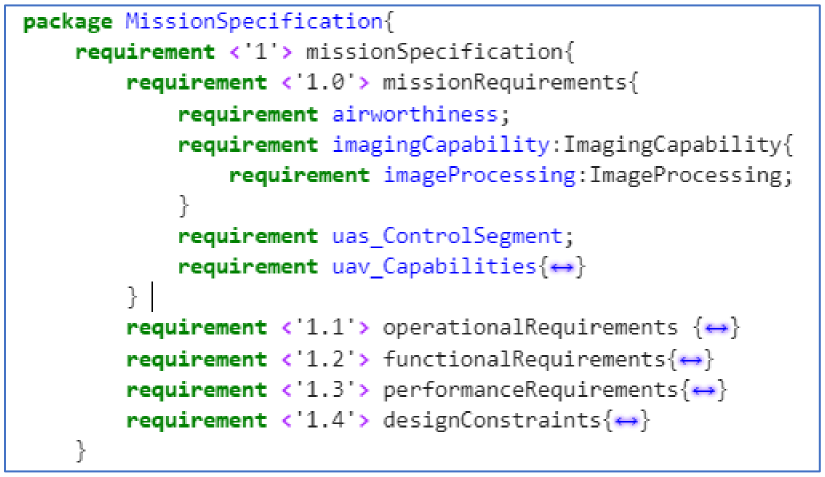
The approach to integrate this behavior involved establishing a consistent action hierarchy and action flow for the 'Non-Combatant Operations - Scenario 0', which was renamed 'Scenario_0'. The Scenario_0 action and its nested actions were referenced as perform actions of the parts that compose the Skyzer mission domain parts hierarchy in **Figure 36**, **Figure 37**, and **Figure 38**. The messages that were transformed from the SysML v1 sequence diagram were then further integrated into this structure as messages between these parts. This resulting behavior between the actions in Scenario_0 performed by the parts and the messages between the parts can be further analyzed for consistency, and additional constraints can be added to ensure the integrated behavior specifies the proper sequence of messages and actions.

This same approach can be applied to other behaviors to ensure the overall model captures the desired integrated behavior consistent with the structure. As a result of this effort, there will no doubt be updates to the use cases, actions, event/message sequences, and states if applicable.

A few potential inconsistencies that surfaced included a missing actor (INMARSAT) from the ‘Activate and Launch UAV’ use case in **Figure 22**. Also, UCARS is part of the Skyzer system rather than a separate actor. There also appear to be some actions missing from the action flow to support the message ‘Load Final Mission'. Potential inconsistencies would need to be reviewed with subject matter experts and the model can be refined based on the results of the review.

4.2.7 Refactor the Requirements

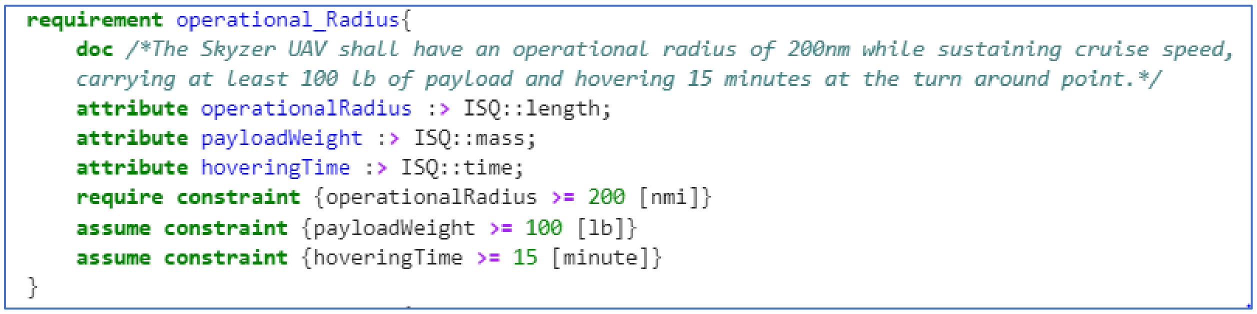
The SysML v1 Skyzer model had many mission level requirements in the Mission Requirements package which were transformed to requirement definitions as described previously. There were several nested packages within the Mission Requirements package that each contained a set of requirements such as the Operational Requirements in **Figure 16**. The refactoring focused on establishing a top-level requirement called requirement <'1'> *missionSpecification* that contained the mission requirements in a single requirements hierarchy to aid in traceability analysis (e.g., requirements allocation, derivation, satisfaction, verification). A partial view of the requirements hierarchy is shown in **Figure 43**.



```
package MissionSpecification{
    requirement <'1'> missionSpecification{
        requirement <'1.0'> missionRequirements{
            requirement airworthiness;
            requirement imagingCapability:ImagingCapability{
                requirement imageProcessing:ImageProcessing;
            }
            requirement uas_ControlSegment;
            requirement uav_Capabilities{↔}
        } |
        requirement <'1.1'> operationalRequirements {↔}
        requirement <'1.2'> functionalRequirements{↔}
        requirement <'1.3'> performanceRequirements{↔}
        requirement <'1.4'> designConstraints{↔}
    }
```

Figure 42. Mission Requirements Hierarchy (Refactored)

This is also an opportunity to leverage the more precise nature of SysML v2 requirements and identify critical requirements that can be specified more formally using requirements constraints. For example, the measure of effectiveness for cruise speed is specified in a requirement that can be expressed more formally as follows:



```
requirement operational_Radius{
    doc /*The Skyzer UAV shall have an operational radius of 200nm while sustaining cruise speed,
    carrying at least 100 lb of payload and hovering 15 minutes at the turn around point.*/
    attribute operationalRadius :> ISQ::length;
    attribute payloadWeight :> ISQ::mass;
    attribute hoveringTime :> ISQ::time;
    require constraint {operationalRadius >= 200 [nmi]}
    assume constraint {payloadWeight >= 100 [lb]}
    assume constraint {hoveringTime >= 15 [minute]}
}
```

Figure 43. Formalizing the Operational Radius Requirement with Constraints

Notice also in **Figure 44,** that a mix of metric and English units are used for the attributes of the requirement that are from the SysML v2 quantities and units' libraries. This is accommodated by the SysML v2 model and a conformant tool can enable units checking and dimensional analysis.

4.2.8 Refactor Requirements Traceability

The requirements traceability can be significantly impacted by the changes to the structure and behavior described previously. An assessment of requirements allocation, satisfaction, verification, derivation, and refinement should be performed to validate they reflect the proper traceability.

For example, the operational radius requirement in **Figure 43** was allocated to a measure of effectiveness (moe) on a part called *missionMOEs* in the transformed SysML v2 model. However, this part was no longer needed in the refactored part structure and now should be re-allocated to be a moe of the *skyzerEnterprise*.

4.2.9 Additional Refactoring

There are many other opportunities beyond what was discussed in this paper to reorganize and refactor the model and leverage SysML v2 capabilities. However, this provides a starting example to build on as the community gains experience with model conversion.

5 Observations and Recommendations

This early effort to manually convert a SysML v1 model to a SysML v2 model is being performed before the availability of commercial SysML v2 modeling tools that will automate part of this process. These results should help set expectations for the effort required, the approach, and the potential benefits of model conversion. However, it should be recognized that these are early observations and are likely to evolve as commercial tools become available and the industry gains experience with model conversion. The observations and recommendations include the following:

1. The conversion steps for transformation post-processing, and validation should be performed incrementally. Performing a batch conversion will make it more difficult to validate the model and will limit the opportunities to significantly improve the model quality and leverage SysML v2 modeling capabilities.
2. The post-processing step should apply the usage-focused paradigm to more fully leverage the SysML v2 modeling capabilities. This will require significant reorganization and refactoring of the model. The model reorganization establishes packages that contain the reusable definition elements that serve as black box specifications with no decomposition. Separate packages are created to contain the mission and system hierarchy that typically include a parts hierarchy, an action hierarchy, and a requirements hierarchy. The usage elements in the hierarchies may be defined by the definition elements as needed to facilitate reuse.
3. Establish a consistent parts hierarchy. The core structure of the SysML v1 model is based on the block decomposition generally starting with a top-level block that serves as a mission context. There may be other implicit structure associated with the OV-1, the swim lanes in activities, the lifelines in sequence diagrams, and the actors in use case diagrams. These implicit structures may not be entirely consistent with the block decomposition. The SysML v2 model provides an opportunity to provide a consistent parts hierarchy from the top-level mission context part down to the lowest level of design.
4. Integrate the behavior with the structure including states, actions, message sequence, and use cases. The initial focus for establishing the integrated behavior is to establish a consistent action decomposition based on the SysML v1 model. There may also be opportunities to create action specializations that share common sets of actions. This has been difficult to do in SysML v1 but is straightforward to do in SysML v2. After the action tree is clearly established, the parts that perform the actions can be established. Although there were no states in the Skyzer model, it is anticipated that the states can then be integrated by identifying which states enable which actions. There were considerable sequence diagrams in the SysML v1 model which were transformed to messages in the SysML v2 model. The messages had to be carefully integrated to ensure they were sent across the correct connections. It is also critical that they be integrated with the action flow, but this may depend on the selected methodology.
5. The post processing may yield significant changes to the system structure and behavior, particularly as it is identified will resolve redundancies, inconsistencies, and other gaps. This, in turn, may impact the requirements allocation/satisfaction and other requirements relationships. In the Skyzer SysML v1 model, there were several requirements that were satisfied by the in the Mission MOE’s block. However, this block was not included in the

SysML v2 model, and a Skyzer Enterprise part was introduced that contained the MOE’s. This change impacted the requirements allocations.

6. There are many opportunities to leverage SysML v2 capabilities to further refine the model and add both precision and expressiveness. An example is the ability to formalize selected requirements with formal constraint expressions that can be evaluated as pass or fail. The standard quantities and unit’s library is much improved over SysML v1 in both precision, expressiveness, and usability. The language adds new concepts to model variability, trade studies, metadata, and many other concepts. SysML v2 also provides the ability to define an alias for any name and can leverage annotations to establish a glossary of terms. There will be new opportunities to integrate with many other applications through the standard API including analysis, configuration management, visualization, and other electrical, mechanical, software, and verification tools.

6 Summary

The transition from SysML v1 to SysML v2 should be carefully planned to include updates to an organization's methods, tools, and training. A particular project should determine when to make the transition to SysML v2 based on near-term, mid-term, and long-term considerations of the benefits, costs, and risks of transition. The timing of the transition on a program should typically be at the start of a new program or system upgrade, with the goal to minimize disruption and maximize the benefit. The program should ensure the proper expertise and resources are available to support their transition to SysML v2 in accordance with the program plan. A program may choose to convert an existing SysML v1 model or start with a new SysML v2 model depending on the state of the SysML v1 model and how well it will support the SysML v2 modeling objectives.

Converting a SysML v1 model to a SysML v2 model includes pre-processing the SysML v1 model, transforming and post-processing the SysML v2 model, and validating that the SysML v2 model adequately reflects the original intent of the SysML v1 model. In addition, the organization should assess the impact to artifacts that were derived or generated from the SysML v1 model and update those artifacts as required. The conversion process should be performed systematically and incrementally, and the results should be validated as part of each increment.

The transformation from a SysML v1 model to a SysML v2 model is anticipated to be enabled by tool automation that implements the SysML v1 to SysML v2 transformation specification. It is also anticipated that pre-processing of the SysML v1 model will be required to remove customizations that are not supported by the standard transformation.

To maximize the advantages of the conversion, it is advisable to reorganize and refactor the transformed SysML v2 model in accordance with the usage-focused paradigm to more fully benefit from the SysML v2 modeling capabilities. If done properly, the additional effort can yield a much more integrated SysML v2 model that is more precise, expressive, regular, interoperable, extensible, and usable than the original SysML v1 model.

7 References

Object Management Group (OMG). (2023, July). About the OMG System Modeling Language Specification Version 2.0 Beta. https://www.omg.org/spec/SysML/2.0/Beta1.

SysML v1 to SysML v2 Transition Community (2024, January 22), SysML v1 to SysML v2 Transition Plan Template (2024, January 22). https://www.omgwiki.org/MBSE/doku.php?id=mbse:sysml_v2_transition:sysml_v1_to_sysml_v2_transition_guidance

Seidewitz, E., and Bajaj, M. (2023, October). *Installation*. GitHub. https://github.com/Systems-Modeling/SysML-v2-Release/tree/master/install/jupyter.

Software Engineering. (Accessed 2024, February). Web page. https://www.cto.mil/sea/swe.

Systems Engineering Research Center (SERC). (2020, May 1). Skyzer IM90-20 Mission Model. https://ime.sercuarc.org/alfresco/mmsapp/mms.html#/projects/PROJECT-ee341bee-eaa7-49be-9f44-e7361699211d/master/documents/_18_5_2_8db028d_1512132621231_771993_109678/views/_18_5_2_8db028d_1512132621231_771993_109678.