**Date**: March 2025

# Satellite Operations Language Metamodel

## Final Specification, version 1.2

| | |
|---|---|
| OMG Document Number: | dtc/2025-03-02 |
| Standard document URL: | http://www.omg.org/spec/SOLM/1.2 |
| Machine Consumable Files: | http://www.omg.org/spec/SOLM/20250301 (XMI File)<br>http://www.omg.org/spec/SOLM/20250203 (Python Library) |


**Primary Contact:**

Justin Boss, Kratos S1, Inc.
email: wkbrd@gmail.com

## USE OF SPECIFICATION – TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

## LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

## PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

## GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means–graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems–without permission of the copyright owner.

## DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE.

IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

## RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph © (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph ©(1) and (2) of the Commercial Computer Software – Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 250 First Avenue, Needham, MA 02494, U.S.A.

## TRADEMARKS

The OMG Object Management Group Logo®, CORBA®, CORBA Academy®, The Information Brokerage®, XMI® and IIOP® are registered trademarks of the Object Management Group. OMG™, Object Management Group™, CORBA logos™, OMG Interface Definition Language (IDL)™, The Architecture of Choice for a Changing World™, CORBAservices™, CORBAfacilities™, CORBAmed™, CORBAnet™, Integrate 2002™, Middleware That's Everywhere™, UML™, Unified Modeling Language™, The UML Cube logo™, MOF™, CWM™, The CWM Logo™, Model Driven Architecture™, Model Driven Architecture Logos™, MDA™, OMG Model Driven Architecture™, OMG MDA™ and the XMI Logo™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

## COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

## ISSUE REPORTING

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page *http://www.omg.org*, under Documents & Specifications, Report a Bug/Issue (http://www.omg.org/technology/agreement.htm).

# Table of Contents

## Table of Contents

# *Revision History*

| Date | Version | Description | Author |
|---|---|---|---|
| November 18th 2005 | 1.0 | Initial version from the submission team. | Brad Kizzort, Jim Cater, Geri Chaudhri |
| April 17th 2006 | 1.1 | Revised submission based on Rhea metamodel | Brad Kizzort, Jim Cater, Geri Chaudhri |
| May 26th 2008 | 1.1.5 | Revised submission based on a service modeling approach | Brad Kizzort |
| August 25th 2008 | 1.2 | Revised submission to address issues raised | Brad Kizzort |
| February 23rd 2008 | 1.3 | Revised submission to incorporate a UML profile approach. | Brad Kizzort, Geri Chaudhri |
| February 22nd, 2010 | 1.4 | Continue refining UML profile approach, adding a system model and an xmi interchange file for the profile. | Brad Kizzort, Geri Chaudhri |
| September 22nd, 2010 | 1.5 | Fully develop mappings to SpacePython and CCL | Brad Kizzort |
| November 8th, 2010 | 1.6 | Add stereotypes and mappings for Verify, Comments, Error Handling | Brad Kizzort |
| December 2nd, 2010 | 1.7 | Remove UML profile, add corrections and clarification based on comments from AB. | Brad Kizzort |
| February 20th, 2011 | 1.8 | Cleanup metamodel based on comments from AB. Make sure all relationship ends are named, attributes and operations are public. Apply metamodel profile in EA. Remove UML Activity Diagram / Profile dependencies and define as standalone MOF metamodel. Provide semantics for all classes. | Brad Kizzort |
| March 18th, 2011 | 1.9 | Remove Java types from model and make private attributes public in the model. Update external spec references. Clarify Time class names, semantics and terminology and introduce concept of execution loci. Update figures affected by model changes. | Brad Kizzort |
| April 25th, 2024 | 2.0 | Updates for 1.1 RTF. Added Python version 3 support. | Justin Boss |
| February 11th, 2025 | 2.1 | Updates for 1.2 RTF. | Justin Boss |

# Introduction

Automation of ground station operations is crucial to efficient and cost-effective spacecraft operations. Automation is achieved when scripted procedures are used to conduct normal operations, such as configuration of ground equipment for a satellite contact, commanding the spacecraft or payload to a new configuration, commanding a spacecraft maneuver, etc. There are a variety of spacecraft operations scripting languages in use today. These languages are incompatible between different ground system developers and spacecraft vendors. Transfer of a satellite from one ground system to another ground system, as would occur during a ground system upgrade, is therefore more expensive due to the required conversion of thousands of lines of automation scripts. A common meta-model format for capturing the operations procedure definition from one implementation and allow conversion into another implementation scripting language would provide significant cost savings, even if 100% conversion is not achieved. Another area of benefit would be direct transfer of test and configuration procedures from spacecraft component test environments into satellite integration environments and into operations environments.

# 1 Scope

## 1.1 General

This specification defines a meta-model, Satellite Operations Language Meta-model (SOLM), for representing spacecraft operations procedures. These procedures contain sequences of instructions to conduct spacecraft operations, typically consisting of spacecraft commands and spacecraft telemetry comparisons. These procedures may also include the configuration of ground equipment, configuration of spacecraft test equipment, execution of ground testing, and execution of on-orbit testing. Historically, these procedures have been captured in flowcharts, text manuals, and a number of different scripting languages used for ground station automation. A standard meta-model to represent spacecraft operations procedures will facilitate the transfer of procedures between the spacecraft vendor and the spacecraft operator, as well as allow for maintenance and transfer of the procedures across different ground systems employed over the lifetime of the spacecraft.

This specification is primarily aimed at providing procedure portability for earth-orbiting satellites. While deep-space spacecraft use similar operational procedures, they also use extensive on-board procedures and there are significant considerations in the representation of time that are not incorporated in this meta-model.

SOLM allows the definition of a platform independent model (PIM) of a spacecraft procedure. The PIM can be mapped into a platform-specific model (PSM) for procedure execution. As shown in Figure 1, the spacecraft operator actor represents the operations group that conducts spacecraft operations. This actor is the primary user of the SOLM repository and maintains spacecraft operations procedures in the repository. The SOLM Repository is shown with a multiplicity association with only one spacecraft, due to dependencies on spacecraft-specific command and telemetry definitions. Because there may be multiple operations groups that share control of the spacecraft either simultaneously or over time, the multiplicity relationship is shown as 1 or more spacecraft operators. The spacecraft integrator/manufacturer may provide an initial set of procedures in the SOLM format or these procedures may be translated into SOLM.

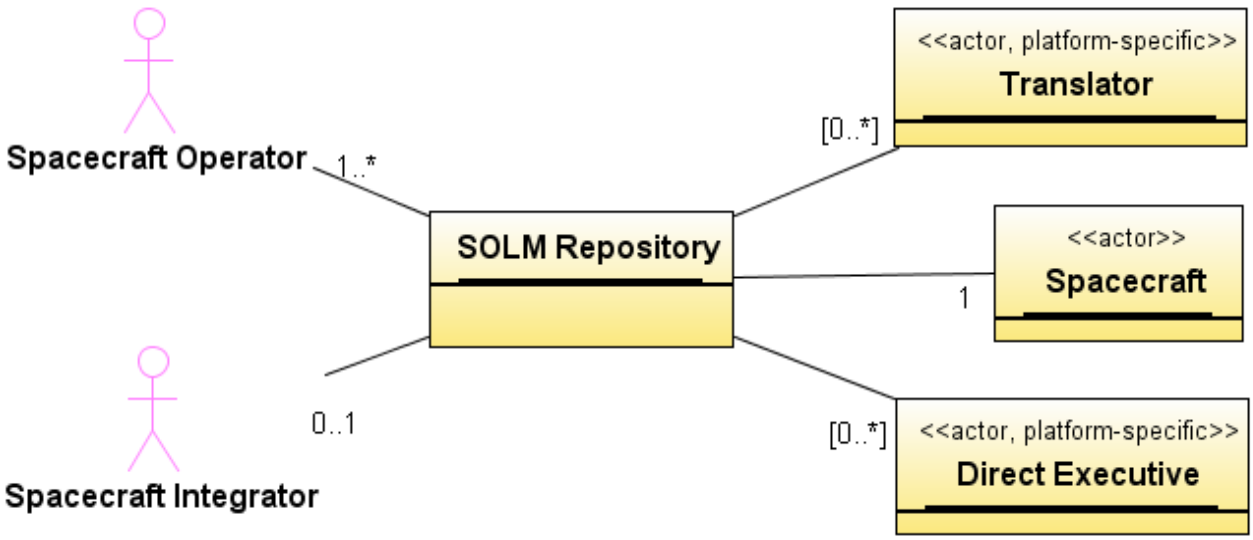


**Figure 1 SOLM Context**

Viewing SOLM as process modeling, SOLM represents the meta-model M2 layer that allows definition of platform independent models (M1) of spacecraft procedures, as shown in Figure 2. Occurrences of procedure executions are the M0 layer, taking on specific parameter values and event times. Spacecraft operators and integrators can develop and exchange M1 models for a specific spacecraft by using SOLM as the common M2 metamodel.

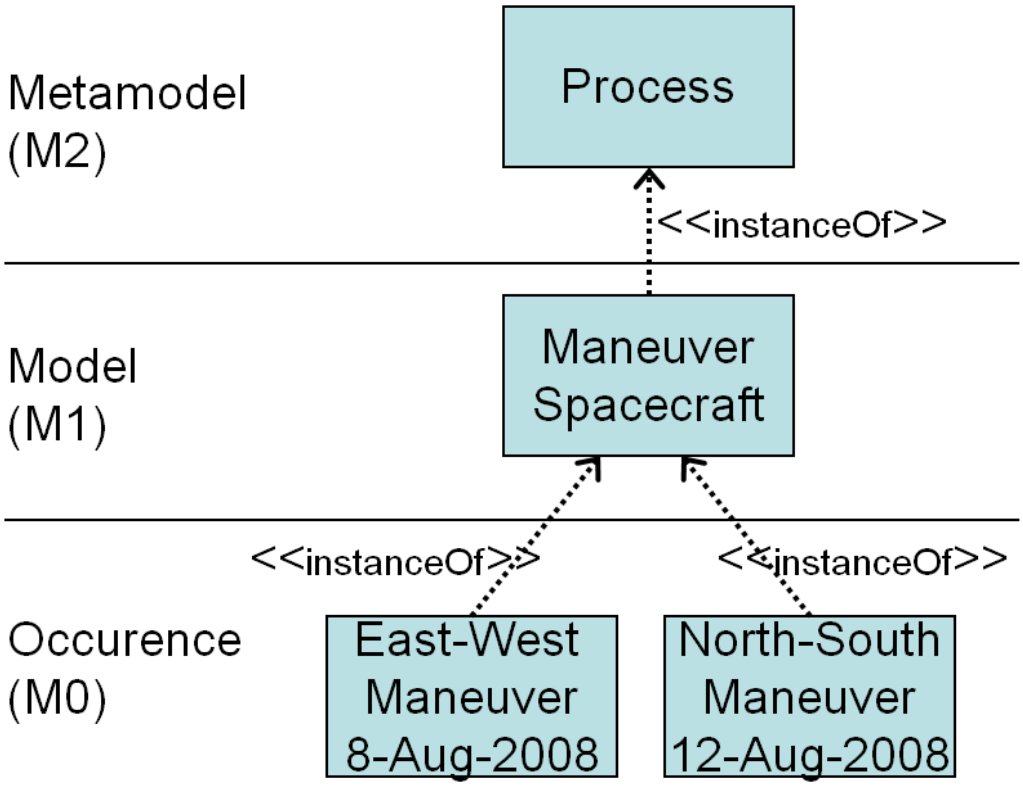


**Figure 2 Process Modeling**

## 1.2 Environments Supporting SOLM

There are two environments that support the SOLM. The first environment is the modeling environment for defining platform independent spacecraft operations procedures, which is represented as the SOLM Repository in Figure 1. The second environment is the platform-specific procedure execution environment, represented by the two secondary system actors, Translator and Direct Executive, in Figure 1. Compliant implementations can provide translation, modeling or execution in either or both of these environments.

A SOLM Translator reads a spacecraft procedure PIM and translates it to a procedure that can be executed by a specific ground system. This procedure will typically be in the native scripting language used by the ground system. Choosing a SOLM Translator as the procedure mapping environment allows for minimal performance impact when implementing SOLM for an existing ground system. The target language for a translator is a Domain Specific Language (DSL), and many DSL's may be targeted by a translator.

A SOLM Modeling Tool maintains a repository of spacecraft procedure PIM's and can export PSM's for a specific execution environment.

A SOLM execution environment is a specific ground system used for spacecraft test or operation. A SOLM-compliant ground system may use a Translator or a Direct Executive to execute procedures provided as SOLM models in an XMI document or as a SpacePython procedure file. The SpacePython language is a DSL that is designed to accept all of the SOLM-defined model in the target procedure.

A SOLM Direct Executive executes the spacecraft procedure reading the PIM directly, without translating it into a specific intermediate representation.

## 1.3 Transition to SOLM

There is a large body of existing spacecraft operations procedures. Initially, the most desirable SOLM capabilities will be translation of existing procedures into a SOLM repository and either translating or direct execution environments for executing SOLM-based procedure models. As the base of SOLM-compliant execution environments expands, and the body of SOLM-based procedure models expands, the market for modeling tools to create, validate, and maintain SOLM-based procedures will develop. A significant part of this specification is the mapping of existing procedure languages to SOLM, and this mapping is intended to be bi-directional to speed the transition. Mappings for other existing or new scripting languages for spacecraft operations may be developed and published as SOLM-related specifications.

# 2 Conformance

A SOLM modeling tool must be able to read and write a spacecraft procedure PIM as an XMI document or as a SpacePython procedure file. A modeling tool must support definition of procedures containing all of the standard SOLM elements, including transformation of XML Telemetry and Command Exchange (XTCE) documents to define the Command and Parameter objects that may be referenced by the procedure model, and transformation of GEMS parameter and directive definition documents into DirectiveTemplate and Parameter objects that may be referenced by the procedure. A modeling environment may also support simulation of procedure

execution and/or an execution display of the procedure in an execution environment. When a SOLM modeling tool is requested to write a spacecraft procedure PSM for a target platform that does not support exception handling or threaded procedure execution defined in the procedure model, it must generate an error message identifying the activity diagram node or procedure file line number that causes the inability to translate.

A compliant SOLM execution environment can provide three levels of compliance:

1. Level 1 compliance must provide execution of procedure models containing conditionals, looping, timed waits, SOLM::NativeProcedure invocations, and Command and Parameter objects from an XTCE document. It must also support exception handling and early termination of a procedure due to errors in a procedure step.
2. Level 2 compliance must provide execution of procedure models as in 1, but also including DirectiveTemplate and Parameter objects from a GEMS equipment definition.
3. Level 3 compliance must provide execution of procedure models as in 2, but also support parallel execution threads in a procedure.

# 3 References

## 3.1 Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply:

The following OMG standards provided the MOF/UML foundation of SOLM:

*UML 2.4.1 (formal/11-08-05, formal/11-08-06)*

*MOF 2.4.1 Core Specification (formal/11-08-07)*

XML Metadata Interchange (XMI) provides a syntactic interchange mechanism for models. It is expected that models conforming to this specification will be interchanged using XMI.

*MOF 2 XMI Mapping, v2.4.1 (formal/11-08-09)*

The following domain specifications provide the formats for the definition of pre-existing objects made available to the procedure modeler in a compliant modeling environment.

*XML Telemetric and Command Exchange (XTCE) Version 1.1* (formal/2008-03-01),

*Ground Equipment Management Specification (GEMS) Version 1.2*, (dtc/2011-04-01)

# 4 Terms and Definitions

For the purposes of this specification, the terms and definitions given in the normative references and the following apply:

**Ground System –** The target system that performs command and telemetry processing for the spacecraft and monitors and controls ground equipment.

**SOLM execution environment** – a software environment for spacecraft operations or testing that executes SOLM procedures either directly or translated into a native format.

**SOLM Modeling Tool** – a software environment that supports the development of spacecraft operations procedures.

# 5 Glossary

**HTTP –** Hyper Text Transfer Protocol

**SOLM** – Satellite Operations Language Metamodel

**UML** – Unified Modeling Language

**URL** – Universal Resource Locator

**XMI** – XML Metadata Interchange

**XSD** – XML Schema Definition

**XSLT** – XML Stylesheet Language Transformations

**XTCE** – XML Telemetry and Command Exchange format

# 6 Meta-model Definition

## 6.1 General

In order to leverage existing commercial standards and technologies, this specification defines a MOF-based meta-model. This approach is intended to allow the application of existing modeling and model transformation environments for spacecraft operations procedures.

The SOLM procedure execution environment could be modeled as interacting with two external actors, an operator, and the ground system that controls the spacecraft and ground equipment, as shown in Figure 3. The procedure execution environment may actually be part of the Ground System, but for the purposes of defining the procedure, the interactions with the operator and the ground system are the significant features.

The notional sequence diagram illustrates that the interactions with the operator are primarily to obtain parameter values and permission to continue with procedure execution. A specific procedure will have specific parameter requirements or may require no interaction with the operator to complete the activity.

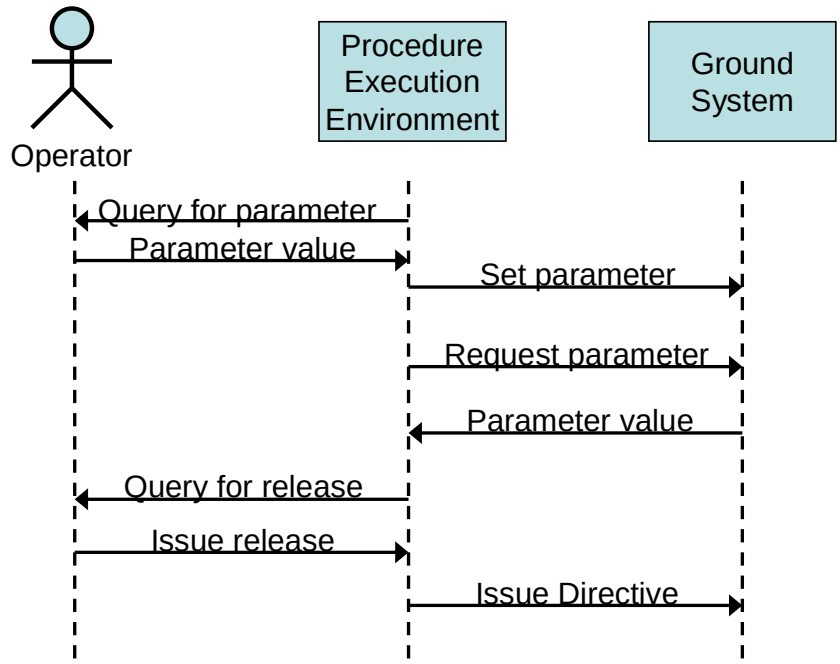


**Figure 3 Notional System Sequence Diagram (Non-Normative)**

The interaction with the Ground System includes parameter values and directives. Parameter values supplied to the procedure can guide or change procedure execution paths or supply values for directives. Parameter values set in the Ground System by the procedure may guide or control execution in the ground system. Directives are an abstraction that covers both spacecraft commands and ground system commands. Directives result in commands being transmitted to the spacecraft or reconfiguration of the Ground System.

This notional sequence diagram also illustrates the critical actions that must be logged by a procedure execution environment as an operations log. Each interaction with the Operator or Ground System is a critical action that must be logged as part of the operations log.

Existing spacecraft procedure definition and development lends itself well to simple flowcharting. Most procedures are simple sequences of commands, with some conditional checks to verify system state before or after a command transmission. Existing languages attempt to make it easy to send spacecraft commands or configure ground equipment, but are not strong in arithmetic or computational performance. The SOLM proposal uses a metamodel similar to the UML Activity Diagram to capture a workflow for the modeled procedure.

Reuse of the activity diagram metamodel implies that a target language may be required to support threading for multi-threaded activities and handling for early/error termination of a procedure or sub-procedure, in order to handle any procedure defined in an activity diagram. Not all existing control languages provide these capabilities, but the SpacePython described in this specification does.

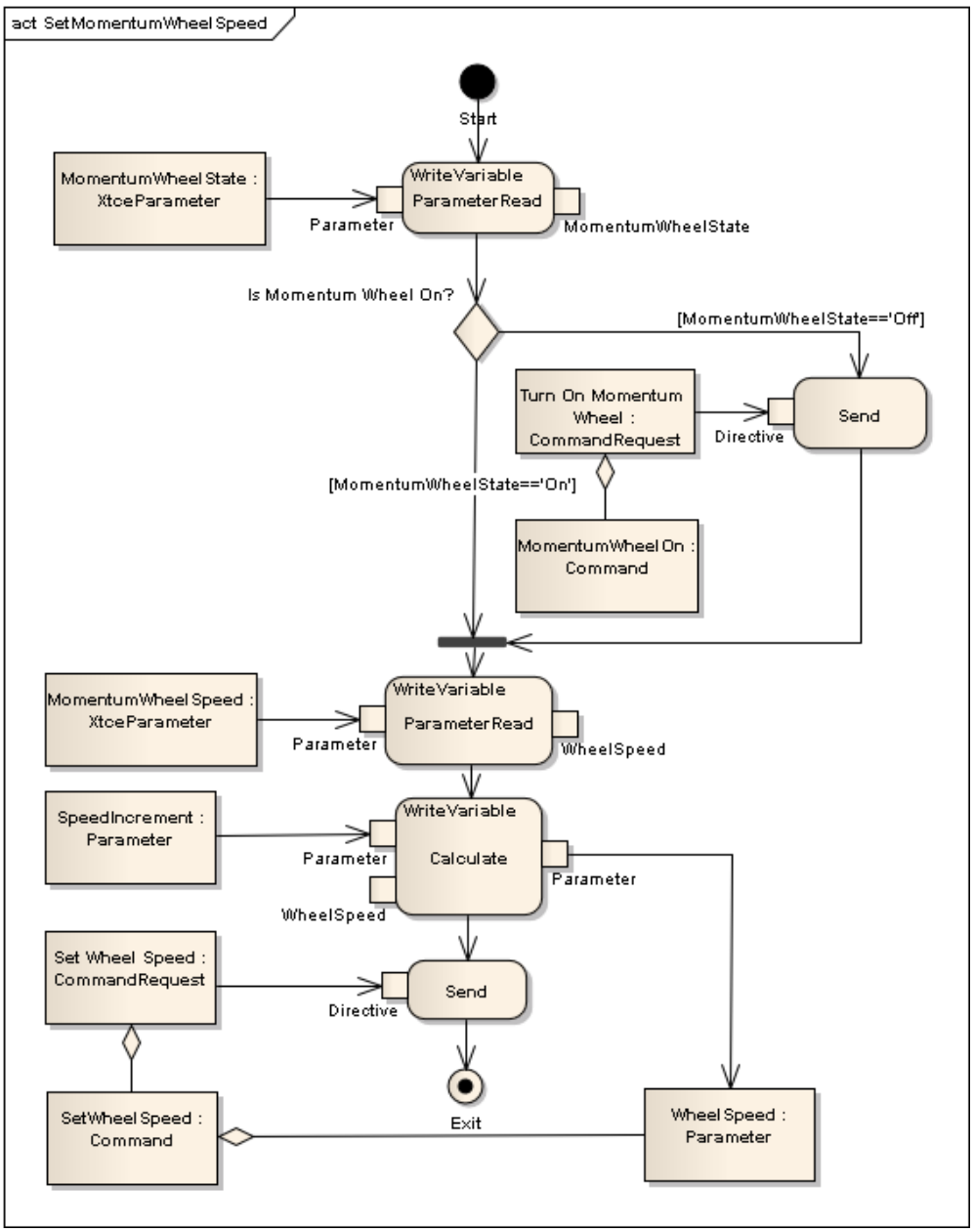


**Figure 4 Notional SOLM Procedure Activity Diagram (Non-Normative)**

Figure 4 illustrates the definition of a simple spacecraft operating procedure using an activity diagram. This procedure will also be used in demonstrating the mapping of SOLM to two existing spacecraft operations languages.

Figure 5 is an overview of the key classes and relationships for SOLM. Each of the following sections will provide subset of the model and descriptions and semantics for the classes. All classes in SOLM are presented as UML in this document, but the normative metamodel is provided as a MOF XMI file. The classes outlined with the dashed XTCE and GEMS box represent classes that will be instantiated from XTCE and GEMS definitions by the SOLM modeling environment.

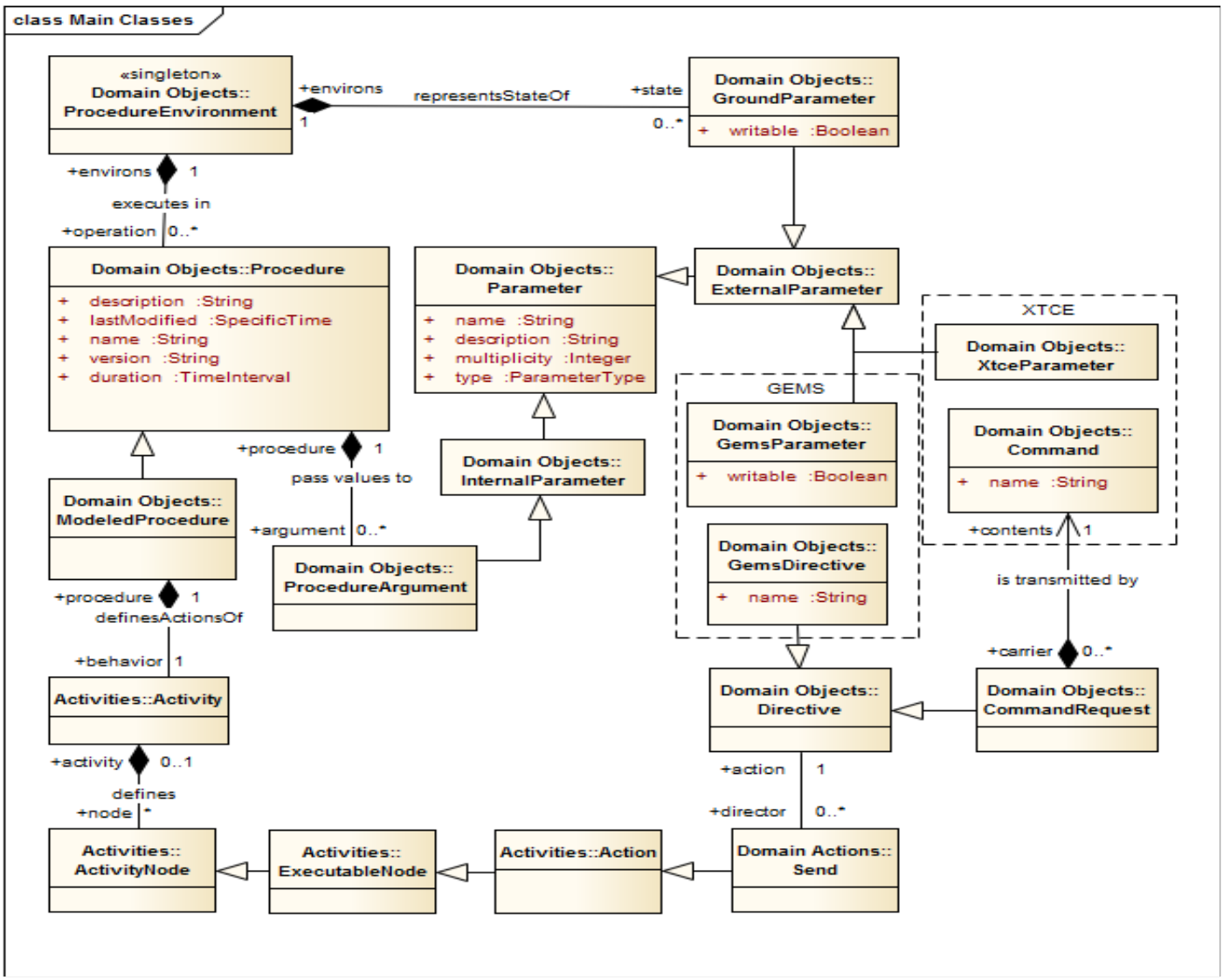


**Figure 5 Key SOLM Classes**

The ProcedureEnvironment maintains a collection of Procedures with specified ProcedureArguments that defines how to invoke each Procedure. The Procedures that are ModeledProcedures have an Activity that defines the procedure step-by-step. The ProcedureEnvironment may also have GroundParameters that are used within Procedures and are treated as global to all procedures, as are the other ExternalParameters that are defined for spacecraft and ground equipment in the XTCE and GEMS definitions. One of the key Actions in an Activite is the Send action, which is causes a Directive to be sent during procedure execution. Directives include GemsDirectives and CommandRequests that are the primary contents of spacecraft operations procedures.

## 6.2 Procedure Invocation

Procedure invocation, either manual or via an Invoke action in a running procedure, is more simplistic than the UML model. For that reason a simplified procedure signature is modeled as shown in Figure 6. All ProcedureArguments are input to the procedure, there is no concept of an out or inout ProcedureArgument. Any outputs use the Send or ParameterWrite action. ProcedureArgument is derived from Parameter and the types match those of the GEMS and XTCE parameters.

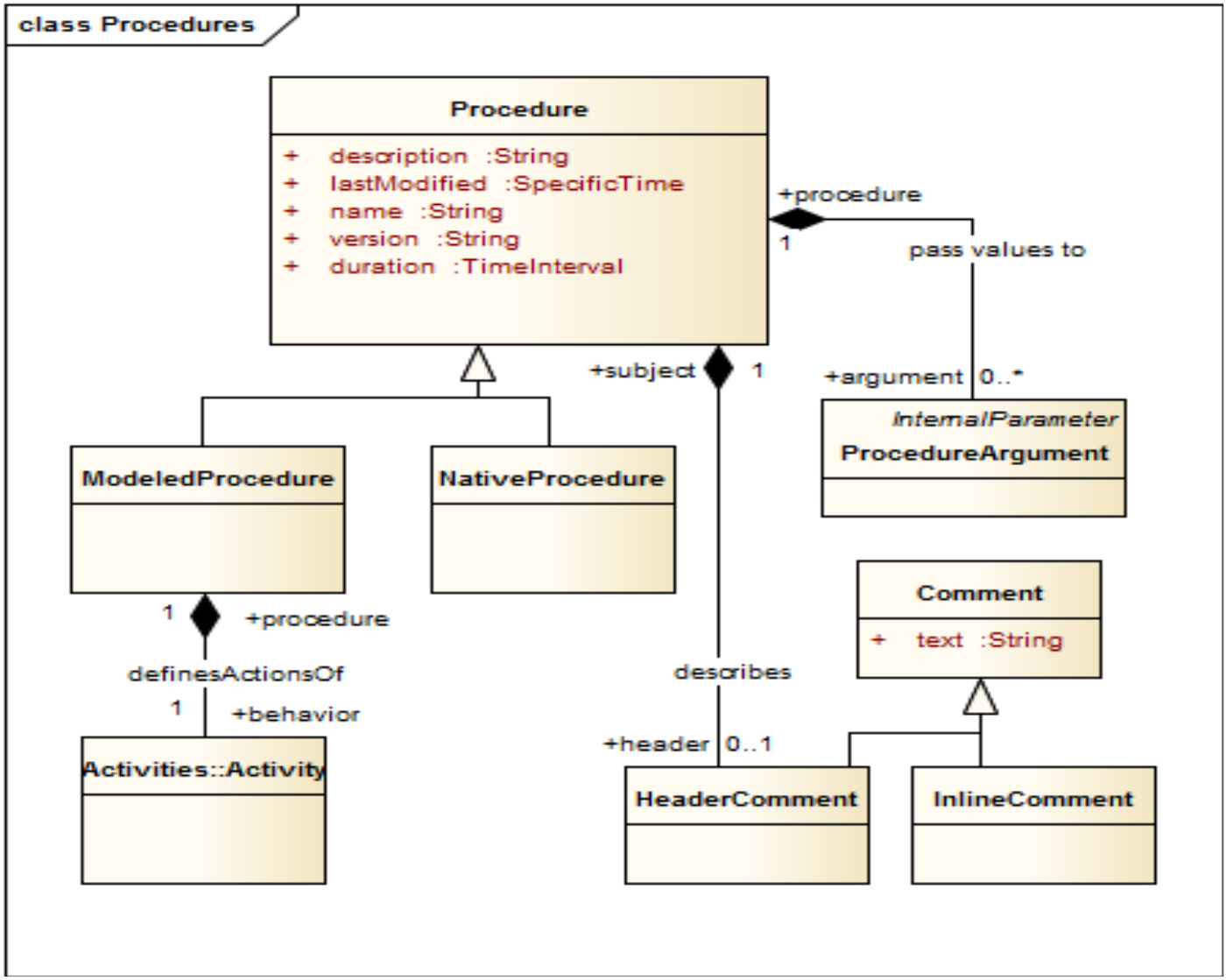


**Figure 6 Procedure Signatures**

There may also be sub-procedures that are in the native scripting language that are not translated into the set of SOLM procedure models shown in the diagram as class NativeProcedure. These sub-procedures must also be able to be invoked from a SOLM model. Because of the different capabilities of different ground system languages in passing and returning values to/from sub-procedures, the sub-procedure invocation is cast as input only Parameters defined in a procedure signature with a single integer returned from the procedure invocation. Additional output Parameters may be set within the procedure only as ExternalParameters.

### 6.2.1 ModeledProcedure

A ModeledProcedure is a Procedure with an Activity definition for all of the operational steps.

#### 6.2.1.1 Generalizations

- Procedure, page 12

#### 6.2.1.2 Attributes

No additional attributes.

#### 6.2.1.3 Associations

- behaviour:Activity – the modelled behaviour of the Procedure.

#### 6.2.1.4 Constraints

No additional constraints

#### 6.2.1.5 Semantics

A ModeledProcedure has one activity definition with a single InitialNode that is the start of execution.

### 6.2.2 NativeProcedure

A NativeProcedure is a Procedure that can be invoked by another Procedure in SOLM, but does not have a modelled behaviour. It is defined in the modelling environment so that it can be invoked by modelled procedures and must be provided as a procedure executable by name in the execution environment. It is intended to support procedures that are in the ground system native format that cannot be fully modelled in SOLM due to system-specific extensions.

#### 6.2.2.1 Generalizations

- Procedure, page 12

#### 6.2.2.2 Attributes

No additional attributes.

#### 6.2.2.3 Associations

No additional associations.

#### 6.2.2.4 Constraints

No additional constraints

#### 6.2.2.5 Semantics

A NativeProcedure runs to completion when invoked before returning control to the invoking Procedure. The NativeProcedure may return an error that can be handled by an exception handler

### 6.2.3 Procedure

#### 6.2.3.1 Generalizations

None

#### 6.2.3.2 Attributes

- description:String – A short text description of the effects of the procedure that can be presented to a modeller or operator for procedure selection. This text will typically be included in the header of a script file.
- duration:TimeInterval – An estimated time period that is required for procedure execution that could be used in planning and scheduling the operation. A negative time value must be used to indicate that the procedure is too variable to predict or has no time estimate.
- lastModified:SpecificTime – the last time that the procedure was modified. This will typically be included as a text comment in the procedure to aid configuration management and anomaly resolution.
- name:String – the name of the procedure

- version:String – a version number assigned to the last modification time.

#### 6.2.3.3 Associations

- argument:ProcedureArgument[0..*] – arguments to the procedure that can receive specific values at execution time to modify the behaviour and actions of the procedure.
- header:HeaderComment[0..1] – an optional long text description of the procedure that can be included in the header of a script file.

#### 6.2.3.4 Constraints

No additional constraints

#### 6.2.3.5 Semantics

A Procedure can be invoked with specific ProcedureArgument values at run-time, either directly by an operator submitting it to the SOLM execution environment, or as a sub-procedure invocation by another executing procedure.

### 6.2.4 ProcedureArgument

A ProcedureArgument provides Parameter values to a specific invocation of a Procedure.

#### 6.2.4.1 Generalizations

- InternalParameter, page 31

#### 6.2.4.2 Attributes

No additional attributes.

#### 6.2.4.3 Associations

- procedure:Procedure – the Procedure this argument supplies values to

#### 6.2.4.4 Constraints

No additional constraints

#### 6.2.4.5 Semantics

The value for a ProcedureArgument can be specified at the time of invocation. The value may be referenced in expressions defined in the Activity. A ProcedureArgument is input only, it may be set within the Procedure behaviour, but the changed value is not returned to a calling Procedure.

### 6.2.5 Comment

A Comment is descriptive text in a Procedure

#### 6.2.5.1 Generalizations

None

#### 6.2.5.2 Attributes

- text:String – the descriptive text

#### 6.2.5.3 Associations

No additional associations.

#### 6.2.5.4 Constraints

No additional constraints

#### 6.2.5.5 Semantics

Comments do not affect the execution of a procedure, but provide a way to capture descriptive text in the comments of an existing script.

### 6.2.6 HeaderComment

#### 6.2.6.1 Generalizations

- Comment, page 13

#### 6.2.6.2 Attributes

No additional attributes.

#### 6.2.6.3 Associations

- subject:Procedure – the Procedure this comment describes.

#### 6.2.6.4 Constraints

No additional constraints

#### 6.2.6.5 Semantics

The text contents of a HeaderComment would normally be included in the header of a translated native scipt by a SOLM translator.

### 6.2.7 InlineComment

#### 6.2.7.1 Generalizations

- Comment, page 13

#### 6.2.7.2 Attributes

No additional attributes.

#### 6.2.7.3 Associations

- subject:ActivityNode – the action/condition/parameter this comment describes.

#### 6.2.7.4 Constraints

No additional constraints

#### 6.2.7.5 Semantics

The text contents of an InlineComment would normally be included inline in the translated native script by a SOLM translator.

## 6.3 Activities

The basic model of a ModeledProcedure is captured in an Activity, which is a collection of ActivityEdges and ActivityNodes. SOLM reuses the abstract syntax of the Activity modeling in UML and Foundational UML, but there are some simplifications to the meta-model and semantics, since SOLM is not intended as a general purpose software metamodel. Figure 7 through Figure 10 show the portions of SOLM related to defining an operations procedure as an Activity.

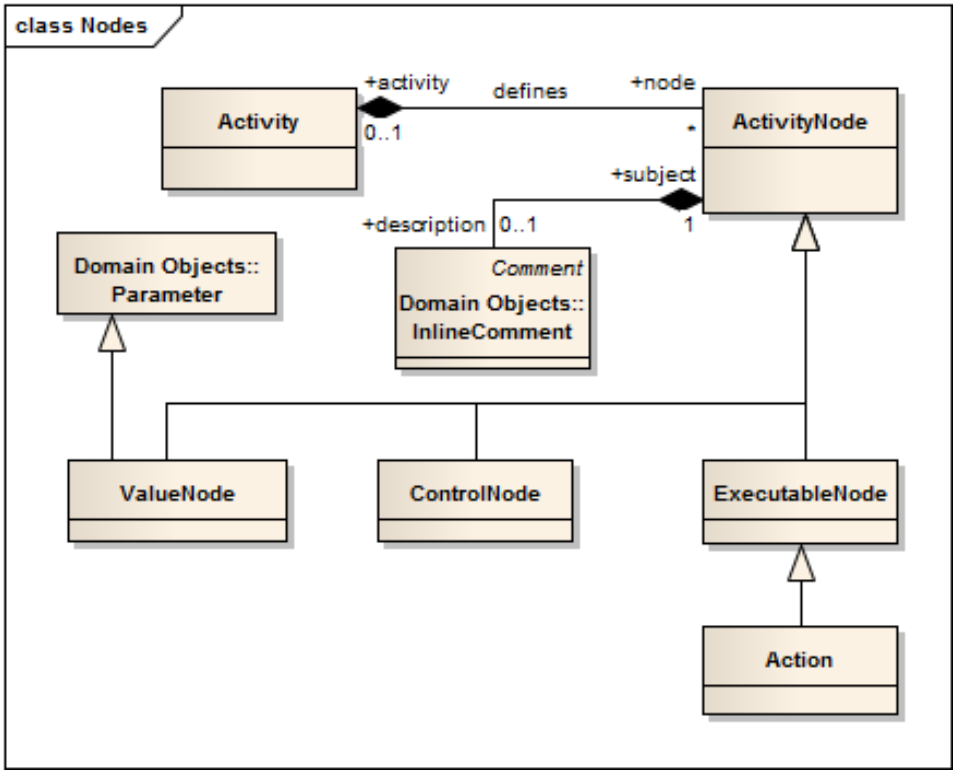


**Figure 7 Activities**

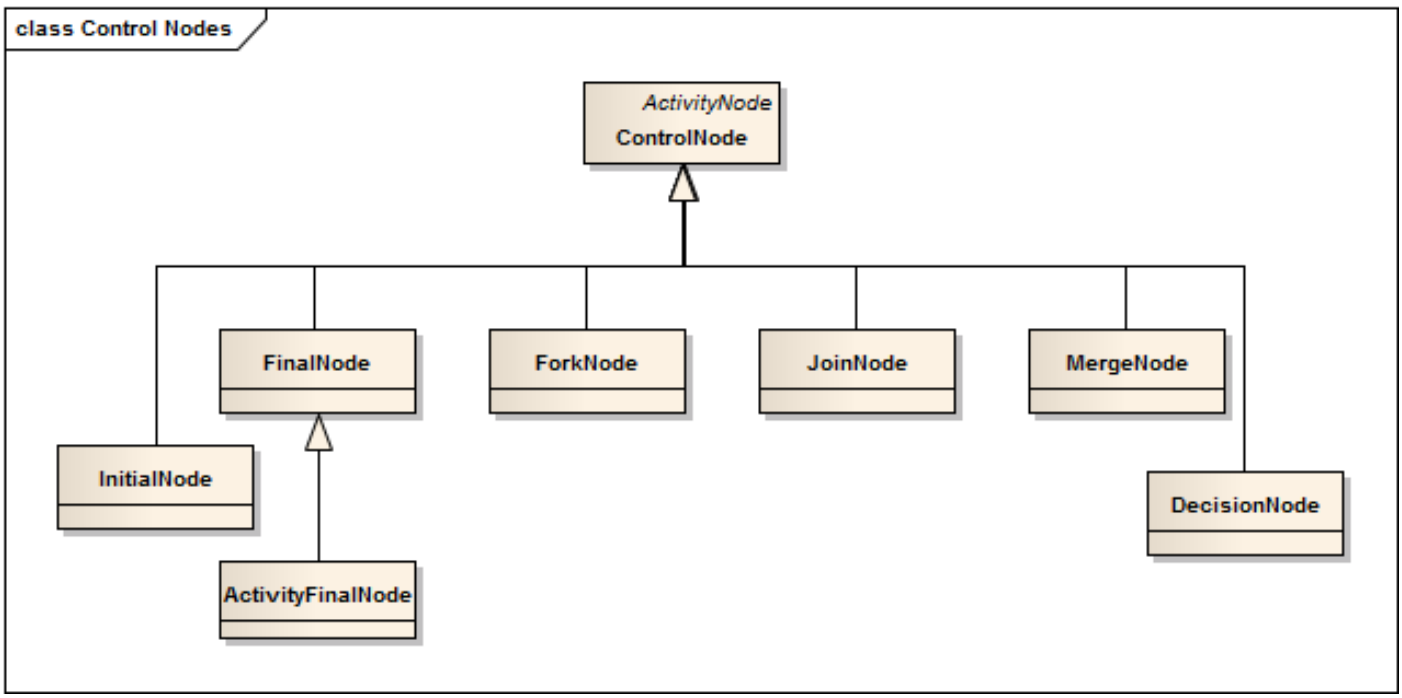


**Figure 8 Control Nodes**

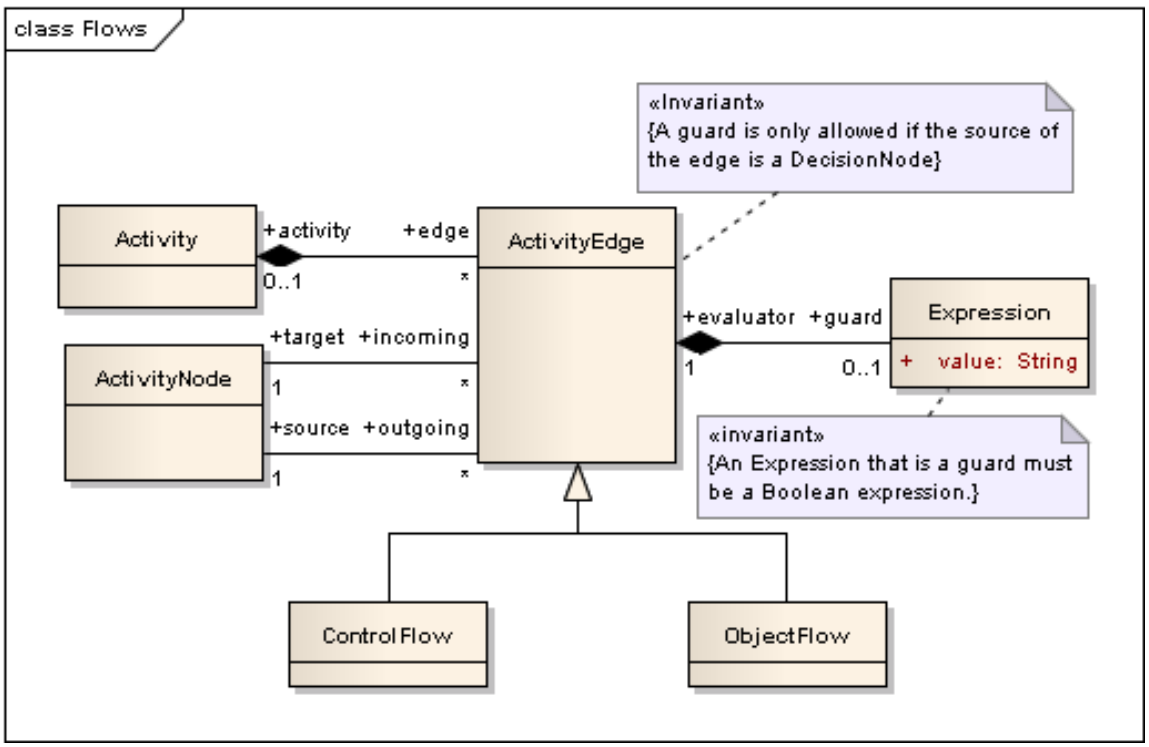


**Figure 9 Control and Data Flow**

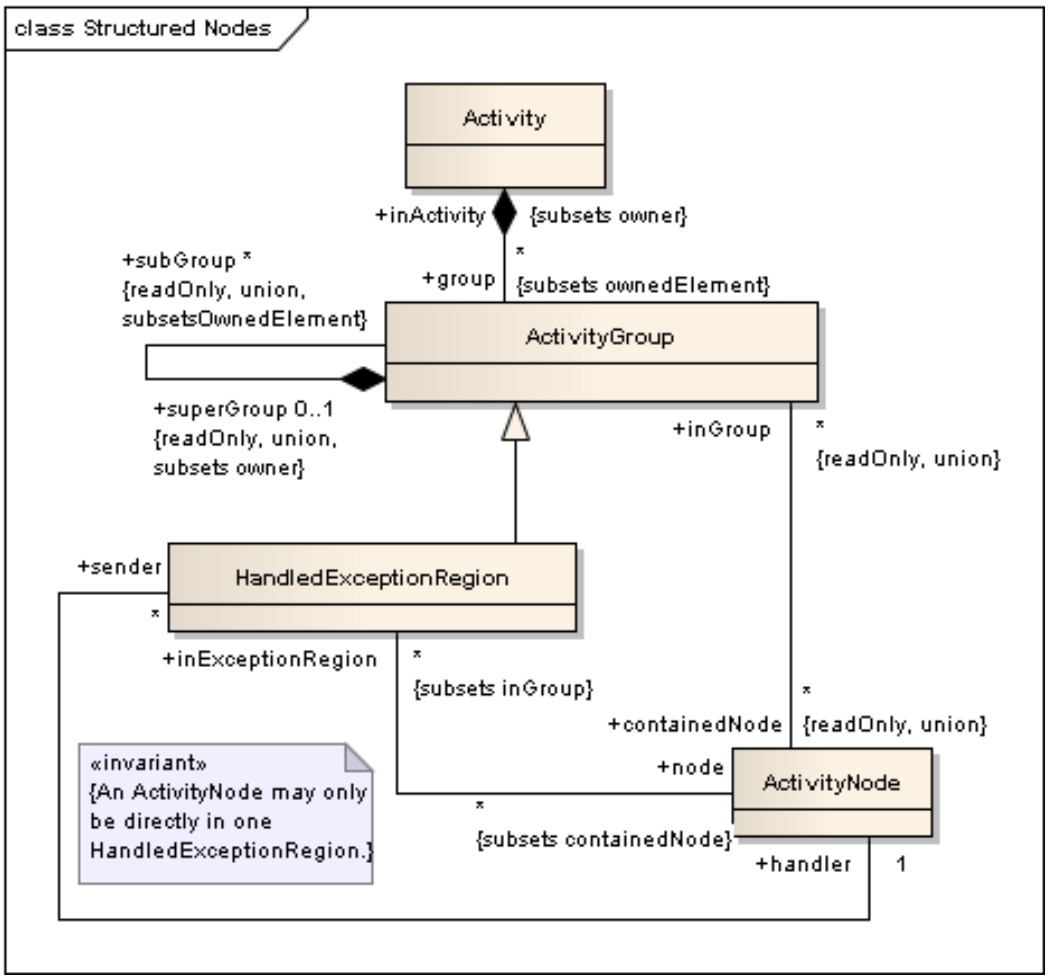


**Figure 10 Exception Handling**

### 6.3.1 Action

An Action is an abstract executable node in the Activity that defines procedure behaviour .

#### 6.3.1.1 Generalizations

- ExecutableNode, page 21

#### 6.3.1.2 Attributes

No additional attributes.

#### 6.3.1.3 Associations

No additional associations.

#### 6.3.1.4 Constraints

No additional constraints

#### 6.3.1.5 Semantics

The sequencing of actions are controlled by control edges within activities, which carry control (see Activity). Except where noted, an action can only begin execution when it has been offered control tokens on all incoming control flows and values are available for all of the incoming object flows. Note

that this differs from the UML semantics in that there are no dynamic object creations and deletions during the activity execution. This is a simplification in keeping with the procedural nature of existing spacecraft operations languages. When the execution of an action is complete, it offers control tokens on its outgoing control flows and specified Parameter values are replaced by outgoing object flows. The steps of executing an action with control and object flow are as follows:

[1] An action execution is created when all its control flow prerequisites have been satisfied (implicit join). Any exceptions to this are listed in the subclass action semantics.

[2] When an action accepts the offers for control tokens, the tokens are removed from the original sources that offered them. If multiple control tokens are available on a single incoming control flow, they are all consumed.

[3] An action continues executing until it has completed. The detailed semantic of execution an action and definition of completion depends on the particular subclass of action.

[4] When completed, an action execution offers control tokens on all its outgoing control flows (implicit fork), and it terminates. Exceptions to this are listed below. The offered tokens may now satisfy the control flow prerequisites for other action executions.

[5] After an action execution has terminated, its resources may be reclaimed by an implementation, but the details of resource management are not part of this specification. All Actions in the current model are locally reentrant. This means that there may be, within any one execution of the containing activity, more than one concurrent execution of the action ongoing at any given time.

### 6.3.2 Activity

An Activity defines the behaviour of a ModeledProcedure

#### 6.3.2.1 Generalizations

None

#### 6.3.2.2 Attributes

No additional attributes.

#### 6.3.2.3 Associations

- edge:ActivityEdge[*] – an edge representing a control or data flow between two ActivityNodes
- node:ActivityNode[*] – the data and control nodes in an Activity
- procedure:ModeledProcedure[1] – the Procedure that the Activity defines the behaviour for.

#### 6.3.2.4 Constraints

No additional constraints

#### 6.3.2.5 Semantics

An Activity has one InitialNode that begins the control flow execution of a Procedure. Activity control flow terminates if an error is encountered, unless the error occurs in a HandledExceptionRegion defined within the Activity.

### 6.3.3 ActivityEdge

An ActivityEdge is an abstract class for directed connections (control or data flow) between two ActivityNodes.

#### 6.3.3.1 Generalizations

None

#### 6.3.3.2 Attributes

No additional attributes.

#### 6.3.3.3 Associations

- activity:Activity[0..1] – the owning activity
- target:ActivityNode[1] – the target node of this flow.
- source:ActivityNode[1] – the source node of this flow.
- guard:Expression[0..1] – an expression limiting control flow on this edge.

#### 6.3.3.4 Constraints

A guard is only allowed if the source of the edge is a DecisionNode

#### 6.3.3.5 Semantics

An ActivityEdge can be either a ControlFlow or an ObjectFlow between two ActivityNodes.

### 6.3.4 ActivityFinalNode

An ActivityFinalNode ends execution of a Procedure.

#### 6.3.4.1 Generalizations

- FinalNode, page 22

#### 6.3.4.2 Attributes

No additional attributes.

#### 6.3.4.3 Associations

No additional associations.

#### 6.3.4.4 Constraints

No additional constraints

#### 6.3.4.5 Semantics

An ActivityFinalNode terminates execution of a procedure. All threads of execution of the procedure terminate. Parallel flows should use a JoinNode or a MergeNode to coalesce with other execution threads prior to the ActivityFinalNode, otherwise active threads associated with parallel flows will be terminated by the thread (control flow) that reaches the ActivityFinalNode.

### 6.3.5 ActivityGroup

An ActivityGroup is a subset of the ActivityNodes in an Activity

#### 6.3.5.1 Generalizations

None

#### 6.3.5.2 Attributes

No additional attributes.

#### 6.3.5.3 Associations

containedNode:ActivityNode[*] – nodes in group

#### 6.3.5.4 Constraints

No additional constraints

#### 6.3.5.5 Semantics

An ActivityGroup is an abstract collection of ActivityNodes. No descendants, other than HandledExceptionGroup is currently defined for SOLM.

### 6.3.6 ActivityNode

An ActivityNode is a data or control node in an Activity.

#### 6.3.6.1 Generalizations

None

#### 6.3.6.2 Attributes

No additional attributes.

#### 6.3.6.3 Associations

- activity:Activity[0..1] – Activity that the node belongs to.
- description:InlineComment[0..1] – description of the ActivityNode effects on procedure.
- incoming:ActivityEdge[*] – incoming data or control flow.
- outgoing:ActivityEdge[*] – outgoiong data or control flow.
- inGroup:ActivityGroup[*] – groups that contain this node
- inExceptionRegion:HandledExceptionRegion[*] – exception handler regions that contain this node.
- sender:HandledExceptionRegion[*] – regions from which this node receives control when exceptions occur.

#### 6.3.6.4 Constraints

No additional constraints

#### 6.3.6.5 Semantics

An ActivityNode is an abstract node. See descendants for semantics.

### 6.3.7 ControlFlow

A ControlFlow transfers execution control from one ActivityNode to the next.

#### 6.3.7.1 Generalizations

- ActivityEdge, page 18

#### 6.3.7.2 Attributes

No additional attributes.

#### 6.3.7.3 Associations

No additional associations.

#### 6.3.7.4 Constraints

No additional constraints

#### 6.3.7.5 Semantics

A control flow is an activity edge that only passes control tokens. Tokens offered by the source node are all offered to the target node.

### 6.3.8 ControlNode

A ControlNode exerts control over the incoming ControlFlow(s).

#### 6.3.8.1 Generalizations

- ActivityNode, page 20

#### 6.3.8.2 Attributes

No additional attributes.

#### 6.3.8.3 Associations

No additional associations.

#### 6.3.8.4 Constraints

No additional constraints

#### 6.3.8.5 Semantics

A ControlNode is an abstract node, see the descendants for specific semantics.

### 6.3.9 DecisionNode

A DecisionNode selects one exclusive outgoing ControlFlow from a set.

#### 6.3.9.1 Generalizations

- ControlNode, page 21

#### 6.3.9.2 Attributes

No additional attributes.

#### 6.3.9.3 Associations

No additional associations.

#### 6.3.9.4 Constraints

- A DecisionNode has one incoming edge and at least one outgoing edge.
- The incoming and outgoing edges must be all control flows.

#### 6.3.9.5 Semantics

Each outgoing ControlFlow from a DecisionNode will have a guard expression, except for a default flow, if one is provided. There is no guaranteed order of guard evaluation, so multiple guards should be exclusive, otherwise any valid guard expression can be chosen during execution and the execution behaviour is unspecified. Only one ControlFlow will be activated by a decision node.

### 6.3.10 ExecutableNode

An ExecutableNode performs domain-specific actions

#### 6.3.10.1 Generalizations

- ActivityNode, page 20

#### 6.3.10.2 Attributes

No additional attributes.

#### 6.3.10.3 Associations

No additional associations.

#### 6.3.10.4 Constraints

No additional constraints

#### 6.3.10.5 Semantics

An ExecutableNode is an abstract action, see the descendants for specific semantics.

### 6.3.11 Expression

An Expression is a Boolean or arithmetic expression consisting of read only references to the Parameters visible at the containing Procedure scope.

#### 6.3.11.1 Generalizations

None

#### 6.3.11.2 Attributes

- value:String – the expression string is a valid Python arithmetic expression.

#### 6.3.11.3 Associations

No additional associations.

#### 6.3.11.4 Constraints

The expression must be a valid Python arithmetic expression. The syntax for Python is maintained in an open-source project at https://docs.python.org/reference.

#### 6.3.11.5 Semantics

An expression is evaluated in a DecisionNode, where the Expression is associated with the outgoing ControlFlows. An expression is also evaluated in the WaitForExpression and ParameterWrite ExecutableNodes.

### 6.3.12 FinalNode

A FinalNode is an abstract node with one descendant, the ActivityFinalNode. No FlowFinalNode is included in SOLM, because it is generally not necessary to terminate and individual flow in an activity, but the FinalNode to ActivityFinalNode inheritance in the UML abstract syntax is retained.

#### 6.3.12.1 Generalizations

- ControlNode, page 21

#### 6.3.12.2 Attributes

No additional attributes.

#### 6.3.12.3 Associations

No additional associations.

#### 6.3.12.4 Constraints

No additional constraints

#### 6.3.12.5 Semantics

See ActivityFinalNode, page 19.

### 6.3.13 ForkNode

A ForkNode starts one or more parallel threads of execution.

#### 6.3.13.1 Generalizations

- ControlNode, page 21

#### 6.3.13.2 Attributes

No additional attributes.

#### 6.3.13.3 Associations

No additional associations.

#### 6.3.13.4 Constraints

No additional constraints

#### 6.3.13.5 Semantics

A ForkNode accepts an incoming control token and places an outgoing control token on each outgoing control flow. This allows the creation of parallel execution paths. These parallel control paths will continue until an ActivityFinalNode or an execution error terminates all control flows.

### 6.3.14 HandledExceptionRegion

A HandledExceptionRegion identifies a set of ActivityNodes that will transfer control to an exception handling ActivityNode, if an error occurs during execution.

#### 6.3.14.1 Generalizations

- ActivityGroup, page 19

#### 6.3.14.2 Attributes

No additional attributes.

#### 6.3.14.3 Associations

handler:ActivityNode – the ActivityNode that will receive control on an exception.

#### 6.3.14.4 Constraints

- An ActivityNode may only be directly contained in one HandledExceptionRegion.

#### 6.3.14.5 Semantics

If an error occurs during the execution of any ExecutableNode or ControlNode in the set of nodes in the region, execution on the thread in error will cease and control will be transferred to the ActivityNode specified as the handler for the HandledExceptionRegion. Parallel control flows will continue. If no HandledExceptionRegion with a handler is defined for the Activity, an error in the execution of any control flow path will cause all control flows to cease execution, as if an ActivityFinal node was executed. ActivityNode containment is constrained so that it is only possible for an error in execution to transfer control to one handler. It is possible to nest HandledExceptionRegions so that an error in the handler for one HandledExceptionRegion may transfer control to the owning region.

### 6.3.15 InitialNode

The InitialNode is where execution begins for an Activity.

#### 6.3.15.1 Generalizations

- ControlNode, page 21

#### 6.3.15.2 Attributes

No additional attributes.

#### 6.3.15.3 Associations

No additional associations.

#### 6.3.15.4 Constraints

No additional constraints

#### 6.3.15.5 Semantics

Only one InitialNode must be specified for an Activity. The InitialNode creates one control token and transfers it on the outgoing ControlFlow.

### 6.3.16 JoinNode

A JoinNode ends parallel threads of execution.

#### 6.3.16.1 Generalizations

- ControlNode, page 21

#### 6.3.16.2 Attributes

No additional attributes.

#### 6.3.16.3 Associations

No additional associations.

#### 6.3.16.4 Constraints

- A JoinNode has exactly one outgoing edge that is a ControlFlow
- All incoming edges are ControlFlows
- There is at least one incoming edge.

.

#### 6.3.16.5 Semantics

If control tokens are offered on all incoming edges, then one control token is offered to the outgoing edge. Multiple tokens offered on the same incoming edge are combined into one.

### 6.3.17 MergeNode

A merge node is a control node that brings together multiple alternate flows. It is not used to synchronize concurrent flows but to accept one among several alternate flows.

#### 6.3.17.1 Generalizations

- ControlNode, page 21

#### 6.3.17.2 Attributes

No additional attributes.

#### 6.3.17.3 Associations

No additional associations.

#### 6.3.17.4 Constraints

- A MergeNode has exactly one outgoing edge.
- A MergeNode has at least one incoming edge.
- All incoming and outgoing edges are ControlFlows.

#### 6.3.17.5 Semantics

All tokens offered on incoming edges are offered to the outgoing edge. There is no synchronization of flows or joining of tokens.

### 6.3.18 ObjectFlow

An ObjectFlow represents a data value being applied to a Parameter, ProcedureArgument, or DirectiveArgument.

#### 6.3.18.1 Generalizations

- ActivityEdge, page 18

#### 6.3.18.2 Attributes

No additional attributes.

#### 6.3.18.3 Associations

- incoming:ActivityNode[1] – the ValueNode or ExecutableNode providing a value.
- outgoing:ActivityNode[1] – the ValueNode or ExecutableNode receiving a value.

#### 6.3.18.4 Constraints

No additional constraints

#### 6.3.18.5 Semantics

If the outgoing node is a ValueNode, the value of the Parameter is set to the value provided by the incoming node. If the outgoing node is an ExecutableNode then the value from the incoming node is provided to the ExecutableNode.

### 6.3.19 ValueNode

A ValueNode is a Parameter that participates in an ObjectFlow within an Activity.

#### 6.3.19.1 Generalizations

- Parameter, page 32
- ActivityNode, page 20

#### 6.3.19.2 Attributes

No additional attributes.

#### 6.3.19.3 Associations

No additional associations.

#### 6.3.19.4 Constraints

No additional constraints

#### 6.3.19.5 Semantics

A ValueNode is a Parameter that participates in an ObjectFlow within an Activity.

## 6.4 Parameters

In order to leverage the XTCE and GEMS specifications already published by the OMG, SOLM has defined Parameters with types that are compatible with both the XTCE and GEMS definitions. A Parameter can be defined in an XTCE document, a GEMS document, or a platform-specific mechanism. In addition to a data type, a parameter may also be restricted in a manner similar to XML schema restrictions. These restrictions should be enforced by the modeling and execution environment for SOLM, but not all target platforms will support all restriction types. Figure 11 illustrates the Parameter relationships for SOLM. Each Parameter may have a CurrentValue, which is the most recent value received from the Space System or ground equipment and an associated timestamp for when the value was generated or received.

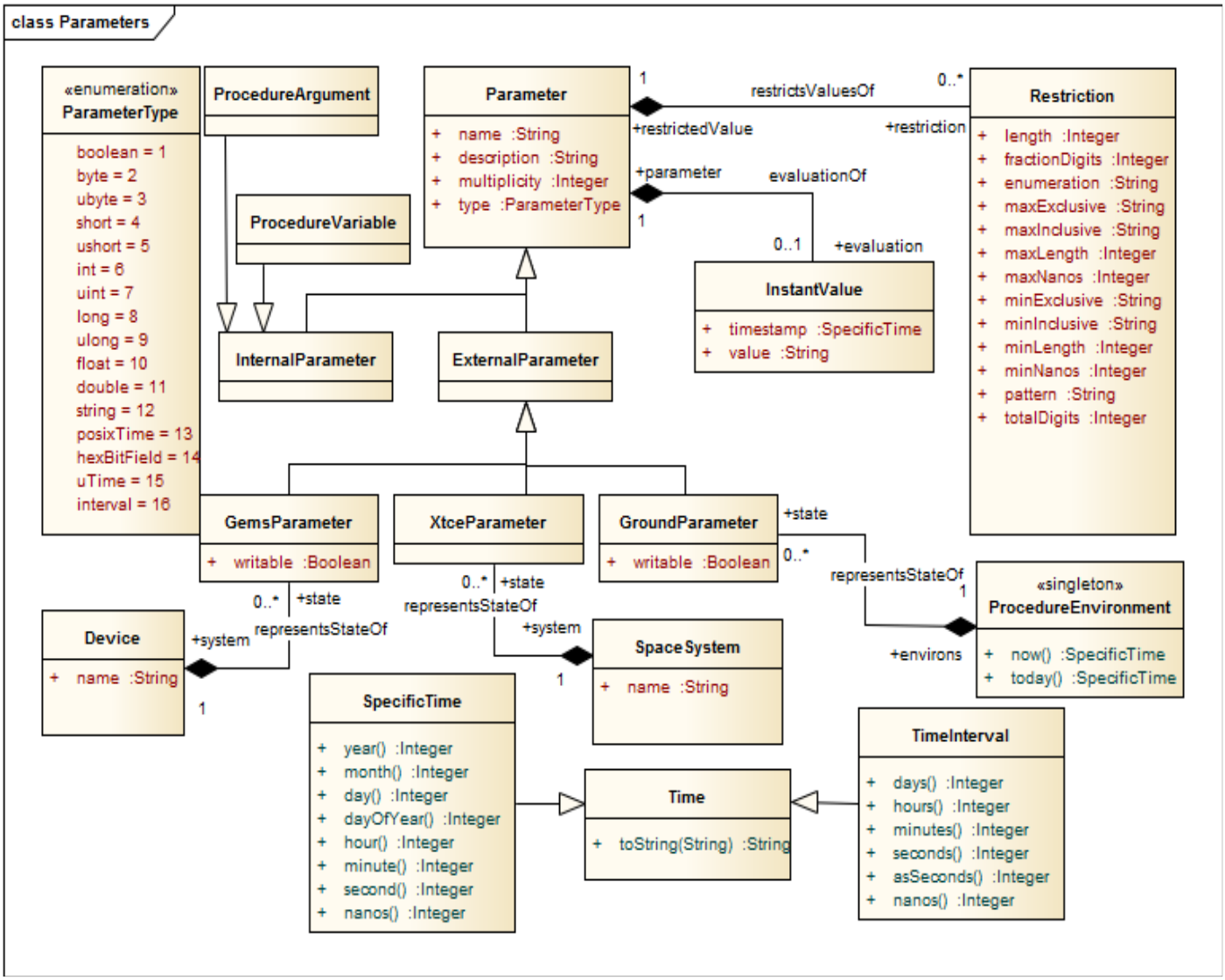


**Figure 11 GEMS and XTCE Parameters**

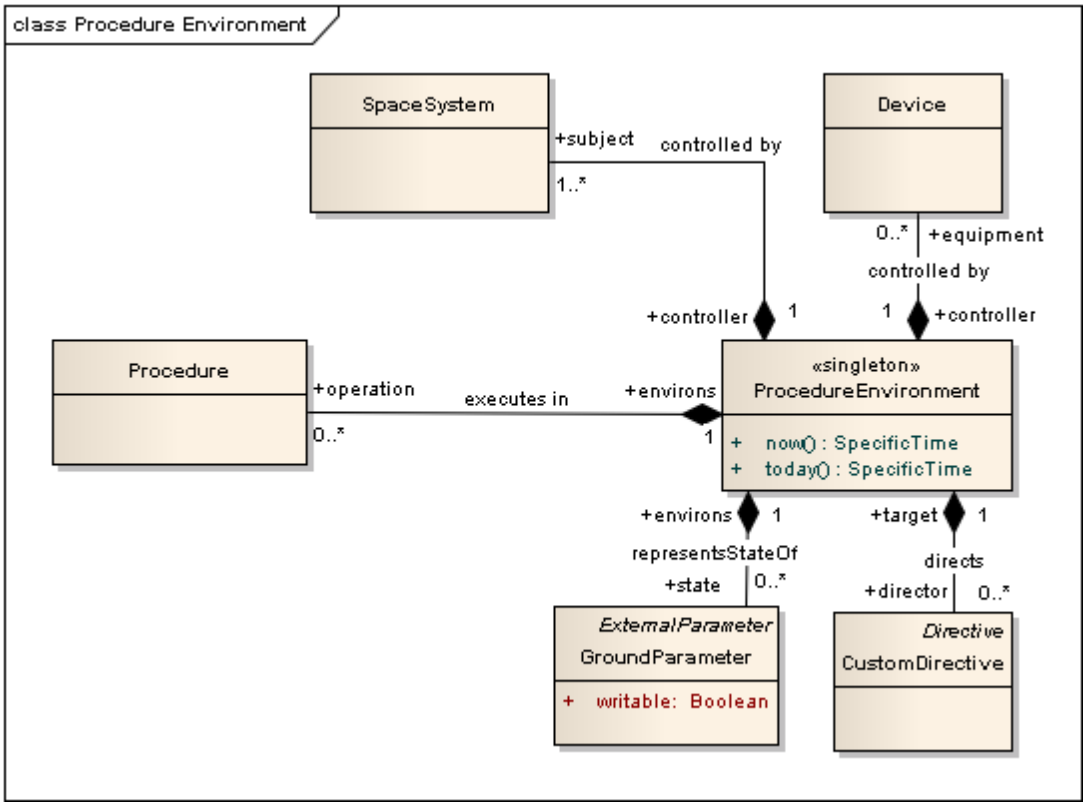


**Figure 12 Procedure Environment**

The SpaceSystem class represents the communications link to the spacecraft or Space System being operated, and the catalog of all XtceParameters represents the current state of the spacecraft. In a similar way, the Device class represents an item of ground equipment in the control system, and the catalog of all GemsParameters for the Device represents the current state of the equipment.

### 6.4.1 SpecificTime

A SpecificTime represents a point in time at the SOLM Execution Environment. This is normally the ground system conducting spacecraft operations. SpecificTime values are not specified within an operations procedure, since that would limit the reusability of the procedure model, but are usually input by the operations team as a ProcedureArgument at invocation, as a result of a Query during execution, or are calculated from a current time obtained from the ProcedureEnvironment at execution time. For portability, the value representation of a SpecificTime will be a String representation of Coordinated Universal Time (UTC), default input format, YYYY-MM-DDTHH:MM:SS.NNN. Other representations or time coordinates required for onboard or remote execution should be calculated within a procedure model, or specified as a ProcedureArgument. Time precision must extend to milliseconds, and future versions of this specification may require micro or nanosecond precision.

#### 6.4.1.1 Generalizations

- Time, page 35

#### 6.4.1.2 Attributes

No additional attributes.

#### 6.4.1.3 Associations

No additional associations.

#### 6.4.1.4 Constraints

No additional constraints

#### 6.4.1.5 Semantics

A SpecificTime may be used as part of a Wait action, in an Expression, as a value for a DirectiveArgument, or as a value for a ParameterWrite. A SpecificTime must be convertible to a POSIX seconds and nanoseconds structure for use with XTCE and GEMS Parameter types. In an expression, the difference between two SpecificTimes is a TimeInterval. A TimeInterval may also be added or subtracted from a SpecificTime to yield another SpecificTime.

#### 6.4.1.6 Operations

- year() returns the Integer value of the Gregorian year associated with the time.
- month() returns the Integer value of the Gregorian month, 1-12.
- day() returns the Integer value of the day of the month, 1-31.
- dayOfYear() returns the Integer value of the ordinal day of the Gregorian year, 1-366.
- hour() returns the Integer value of the hour of the day, 0-23.
- minutes() returns the Integer value of the minute of the hour, 0-59.
- seconds() returns the Integer value of the second of the minute, 0-59.
- nanos() returns the Integer value of the nanoseconds of the second, 0-999,999,999.

### 6.4.2 Device

A GEMS device has a set of Parameters that can be read and written and may also support GemsDirectives to change the Device configuration. A GEMS device is usually ground equipment that is part of the spacecraft ground support.

#### 6.4.2.1 Generalizations

None

#### 6.4.2.2 Attributes

- name:String – unique name for the device.

#### 6.4.2.3 Associations

- state:GemsParameter[0..*] – the set of Parameters that represent the state of the device, some of which may also be settable to change the configuration of the device.
- director:GemsDirective[0..*] – the set of directives that can be used to change device configuration or state.
- controller:ProcedureEnvironment[1] – the ProcedureEnvironment in the ground system that controls this Device.

#### 6.4.2.4 Constraints

No additional constraints

#### 6.4.2.5 Semantics

A Device may be configured by writing GemsParameter values and issuing GemsDirectives associated with the device. Device state may also be used in a procedure by reading GemsParameter values. The

unique name of a Device will be used by the procedure environment to establish a control and status connection to the device.

### 6.4.3 ExternalParameter

An ExternalParameter differentiates Parameters that are external to the Procedure. ExternalParameters may be GemsParameters, XtceParameters, or GroundParameters.

#### 6.4.3.1 Generalizations

- Parameter, page 32

#### 6.4.3.2 Attributes

No additional attributes.

#### 6.4.3.3 Associations

No additional associations.

#### 6.4.3.4 Constraints

No additional constraints

#### 6.4.3.5 Semantics

The ParameterRead action on an ExternalParameter must cause special processing to obtain a new value. A ParameterWrite action an ExternalParameter must change the state of the procedure environment or devices associated with the procedure environment. Because ExternalParameters are, by definition, external to the Procedure, they act like global parameters.

### 6.4.4 GemsParameter

A GemsParameter represents part of the state of a GEMS Device. It has a specific value which may be read from the device. Some GemsParameters are writable, and the configuration of the Device will be changed by a ParameterWrite with an outgoing ObjectFlow to the GemsParameter.

#### 6.4.4.1 Generalizations

- ExternalParameter, page 30

#### 6.4.4.2 Attributes

- writable:Boolean – indicates whether the GEMS device supports setting the value of the Parameter.

#### 6.4.4.3 Associations

- system:Device – the GEMS device containing the Parameter.

#### 6.4.4.4 Constraints

- A GemsParameter with a False writable attribute value cannot be the target of an ObjectFlow.

#### 6.4.4.5 Semantics

See description above.

### 6.4.5 GroundParameter

A GroundParameter represents part of the state of the ground system providing the Procedure environment. It has a specific value which may be read from the system. Some GroundParameters are writable, in which case, setting the value changes the configuration of the ground system.

#### 6.4.5.1 Generalizations

- ExternalParameter, page 30

#### 6.4.5.2 Attributes

- writable:Boolean – indicates whether the ground system supports setting the value of the Parameter.

#### 6.4.5.3 Associations

No additional associations.

#### 6.4.5.4 Constraints

- A GroundParameter with a False writable attribute value cannot be the target of an ObjectFlow.

#### 6.4.5.5 Semantics

See description above.

### 6.4.6 InstantValue

A value with a timestamp for a Parameter.

#### 6.4.6.1 Generalizations

None

#### 6.4.6.2 Attributes

- value:String – the value of the Parameter. The value should be expressed appropriate to the ParameterType and any Restrictions on the ParameterType
- timestamp:AbsoluteTime – the time when the value was sampled or calculated.

#### 6.4.6.3 Associations

- parameter:Parameter – the valued parameter.

#### 6.4.6.4 Constraints

No additional constraints

#### 6.4.6.5 Semantics

Provides a value for a Parameter to be used in expressions or Procedure invocations.

### 6.4.7 InternalParameter

An InternalParameter is a Parameter internal to the Procedure. An InternalParameter may be a ProcedureVariable or a ProcedureArgument.

#### 6.4.7.1 Generalizations

- Parameter, page 32

#### 6.4.7.2 Attributes

No additional attributes.

#### 6.4.7.3 Associations

No additional associations.

#### 6.4.7.4 Constraints

No additional constraints

#### 6.4.7.5 Semantics

The effect of reading or writing an InternalParameter is limited to the Procedure itself. Writing the value of an InternalParameter will cause all later references to the InternalParameter to use the new value.

### 6.4.8 Parameter

A Parameter has a type and a value and is read and/or written by Procedures.

#### 6.4.8.1 Generalizations

None

#### 6.4.8.2 Attributes

No additional attributes.

#### 6.4.8.3 Associations

- evaluation:InstantValue[0..1] – value of the Parameter at an instant in time. If the Parameter has never been reported or calculated it may not have an InstantValue. An Expression that uses a Parameter with no defined Instantvalue causes an exception in the execution that may be handled in a HandledExceptionRegion.
- restriction:Restriction[0..*] – the values that Parameter may take are restricted by the ParameterType and may be additionally restricted by defined Restrictions.

#### 6.4.8.4 Constraints

No additional constraints

#### 6.4.8.5 Semantics

In order to be consistent with XTCE and GEMS Parameters, all values used in SOLM are based on the Parameter class, and have ParameterTypes and Restrictions that are consistent with those specifications. Distinguishing between internal and external Parameter types allows special actions to occur when ExternalParameters are read or written.

### 6.4.9 ParameterType

#### 6.4.9.1 Generalizations

None

#### 6.4.9.2 Attributes

The type enumeration in SOLM is consistent with the types supported by GEMS, XTCE, and most spacecraft operations scripting languages.

#### 6.4.9.3 Associations

No additional associations.

#### 6.4.9.4 Constraints

No additional constraints

#### 6.4.9.5 Semantics

The ParameterType constrains the allowable values and allowable Restrictions for a Parameter.

### 6.4.10 TimeInterval

A TimeInterval represents a negative or positive interval of time. A TimeInterval can be added or subtracted from a SpecificTime to create a new SpecificTime that is later or earlier than the original SpecificTime. The default value representation of a TimeInterval is sPDTHH:MM:SS.NNNNNNNNN, where ‘s’ is an optional ‘+’ or ‘-‘, and D represents as many digits of an integer number of 24-hour days as are necessary or the digit ‘0’. The precision must extend to nanoseconds.

#### 6.4.10.1 Generalizations

- Time, page 35

#### 6.4.10.2 Attributes

No additional attributes.

#### 6.4.10.3 Associations

No additional associations.

#### 6.4.10.4 Constraints

No additional constraints

#### 6.4.10.5 Semantics

A TimeInterval is the result of an Expression taking the difference between to SpecificTimes. An Expression may also calculate a new SpecificTime by adding or subtracting a TimeInterval to/from a SpecificTime. A TimeInterval may be initialized from a POSIX time structure in a GEMS or XTCE Parameter type. It may also be initialized from an Integer number of seconds or a floating point decimal number with fractional seconds. A TimeInterval may be used in a Wait action.

#### 6.4.10.6 Operations

- days() returns the Integer number of whole days in this interval.
- hours() returns the Integer number of whole hours, not including any whole days.
- minutes() returns the Integer number of whole minutes, not including any whole hours.
- seconds() returns the Integer number of whole seconds, not including any whole minutes.
- nanos() returns the Integer number of nanoseconds, not including any whole seconds.
- asSeconds() returns the Integer number of seconds in the entire interval.

### 6.4.11 ProcedureEnvironment

#### 6.4.11.1 Generalizations

None

#### 6.4.11.2 Attributes

No additional attributes.

#### 6.4.11.3 Associations

- director:CustomDirective[0..*] – A collection of CustomDirectives that are part of the procedure environment. These Directives may be defined by a modeling environment to support ground system-specific directives.
- equipment:Device[0..*] – GEMS devices that are part of the ground system.
- state:GroundParameter[0..*] – Collection of Parameters that are specific to a ground system.
- operation:Procedure[0..*] – Collection of Procedures defined for a ground system.

- subject:SpaceSystem[1..*] – Collection of at least one SpaceSystem that is monitored and controlled by the ground system.

#### 6.4.11.4 Constraints

No additional constraints

#### 6.4.11.5 Semantics

The ProcedureEnvironment is a singleton that contains all of the definitions associated with Procedure development for a specific SpaceSystem.

### 6.4.12 ProcedureVariable

#### 6.4.12.1 Generalizations

- InternalParameter, page 31.

#### 6.4.12.2 Attributes

No additional attributes.

#### 6.4.12.3 Associations

- incoming:ObjectFlow[0..*] – source of new value from Procedure action.

#### 6.4.12.4 Constraints

No additional constraints

#### 6.4.12.5 Semantics

A ProcedureVariable is a Parameter with a local procedure scope. Setting the value of the ProcedureVariable from an Action in the procedure Activity definition will have no effect on the ground system or the execution of other procedures.

### 6.4.13 Restriction

A Restriction restricts the allowed values of a Parameter

#### 6.4.13.1 Generalizations

None

#### 6.4.13.2 Attributes

- enumeration:String – restricts the values of ParameterTypes with a String value to the specific list of strings.
- fractionDigits:Integer – restricts the number of digits after the decimal place in a ParameterType with a floating point value.
- length:Integer – restricts the length of a string ParameterType to a specific, exact length.
- maxExclusive:String – restricts the value of a numeric ParameterType, floating point, integer, or the seconds portion of a time type, to be less than the specified value.
- maxInclusive:String – restricts the value of a numeric ParameterType, floating point, integer, or the seconds portion of a time type, to be less than or equal to the specified value.
- maxLength:Integer – restricts the length of a string ParameterType to a maximum number of characters.
- maxNanos:Integer – restricts the nanoseconds portion of a time ParameterType to be less than or equal to the specified value.

- minExclusive:String – restricts the value of a numeric ParameterType, floating point, integer, or the seconds portion of a time type, to be greater than the specified value.
- minInclusive:String – restricts the value of a numeric ParameterType, floating point, integer, or the seconds portion of a time type, to be greater than or equal to the specified value.
- minLength:Integer – restricts the length of a string ParameterType to a minimum number of characters.
- minNanos:Integer – restricts the nanoseconds portion of a time ParameterType to be greater than or equal to a specified value.
- pattern:String – restricts the value of a string ParameterType to the pattern defined by a regular expression.
- totalDigits:Integer – restricts the total number of digits allowed in a floating point ParameterType.

#### 6.4.13.3 Associations

No additional associations.

#### 6.4.13.4 Constraints

The value of specific attributes of the Restriction are constrained to be compatible with the associated Parameter.

#### 6.4.13.5 Semantics

Writing a value to a Parameter or supplying an argument to a ProcedureArgument or DirectiveArgument that does not meet the associated restriction criteria will result in an error that halts script execution unless a HandledExceptionRegion is defined for the ActivityNode where the error occurred. The Query action should limit accepted data to the values allowed by the associated Parameter.

### 6.4.14 SpaceSystem

A SpaceSystem represents a link to the spacecraft under control by the Procedure. The name is taken from the XTCE specification.

#### 6.4.14.1 Generalizations

None

#### 6.4.14.2 Attributes

- name:String – unique name for the SpaceSystem used by the ProcedureEnvironment to establish a link to the SpaceSystem.

#### 6.4.14.3 Associations

- state:XtceParameter[0..*] – the set of Parameters that represent the state of the spacecraft.
- instruction:Command[0..*] – the set of commands that can be used to change spacecraft configuration or state.
- controller:ProcedureEnvironment[1] – the ProcedureEnvironment in the ground system that controls this SpaceSystem.

#### 6.4.14.4 Constraints

No additional constraints

#### 6.4.14.5 Semantics

The SpaceSystem is typically defined by an XTCE document, which results in a set of XtceParameter and Command instances which can be used in the Procedure definition. The unique name of the SpaceSystem is used at procedure execution time to establish a connection with the SpaceSystem being controlled.

### 6.4.15 Time

#### 6.4.15.1 Generalizations

None

#### 6.4.15.2 Attributes

None

#### 6.4.15.3 Associations

No additional associations.

#### 6.4.15.4 Constraints

None.

#### 6.4.15.5 Semantics

A Time is an abstract time value. It represents either a TimeInterval or a SpecificTime.

#### 6.4.15.6 Operations

- toString(String) returns the Time value formatted according to the specified format String or the default format if the String is empty. The format string follows the Python time module specification for time formatting.

### 6.4.16 XtceParameter

An XtceParameter represents a part of the state of a SpaceSystem under control.

#### 6.4.16.1 Generalizations

- ExternalParameter, page 30.

#### 6.4.16.2 Attributes

No additional attributes.

#### 6.4.16.3 Associations

No additional associations.

#### 6.4.16.4 Constraints

No additional constraints

#### 6.4.16.5 Semantics

The telemetry of a SpaceSystem is normally received, calculated, and buffered by the ground system and made available to the Procedure as last reported values, therefore a ReadParameter action does not usually cause communication to the SpaceSystem, merely retrieval of the last reported value. Likewise, setting the value of an XtceParameter does not result in communication with the SpaceSystem under control, but usually results in the update of a derived (non-telemetered) state value, or is simply overwritten when the ground system updates the current value of the Parameter.

## 6.5 Command Transmission

SOLM requires a standard way to invoke the transmission of a command defined in an XTCE document. The modeling environment creates a collection of Command instances based on the MetaCommands defined in the XTCE document for the spacecraft. In order to transmit a command, a CommandRequest is created for a specific Command instance in the modeling environment catalog, and describes how the Command must be handled when transmitted through the link. Parameters required or optionally allowed for the Command are specified in association with the command. These relationships are shown in Figure 13. A GemsDirective is effectively a command to an item of ground equipment and the Command and the GemsDirective are generalized as a Directive which is a single step in an operations procedure.

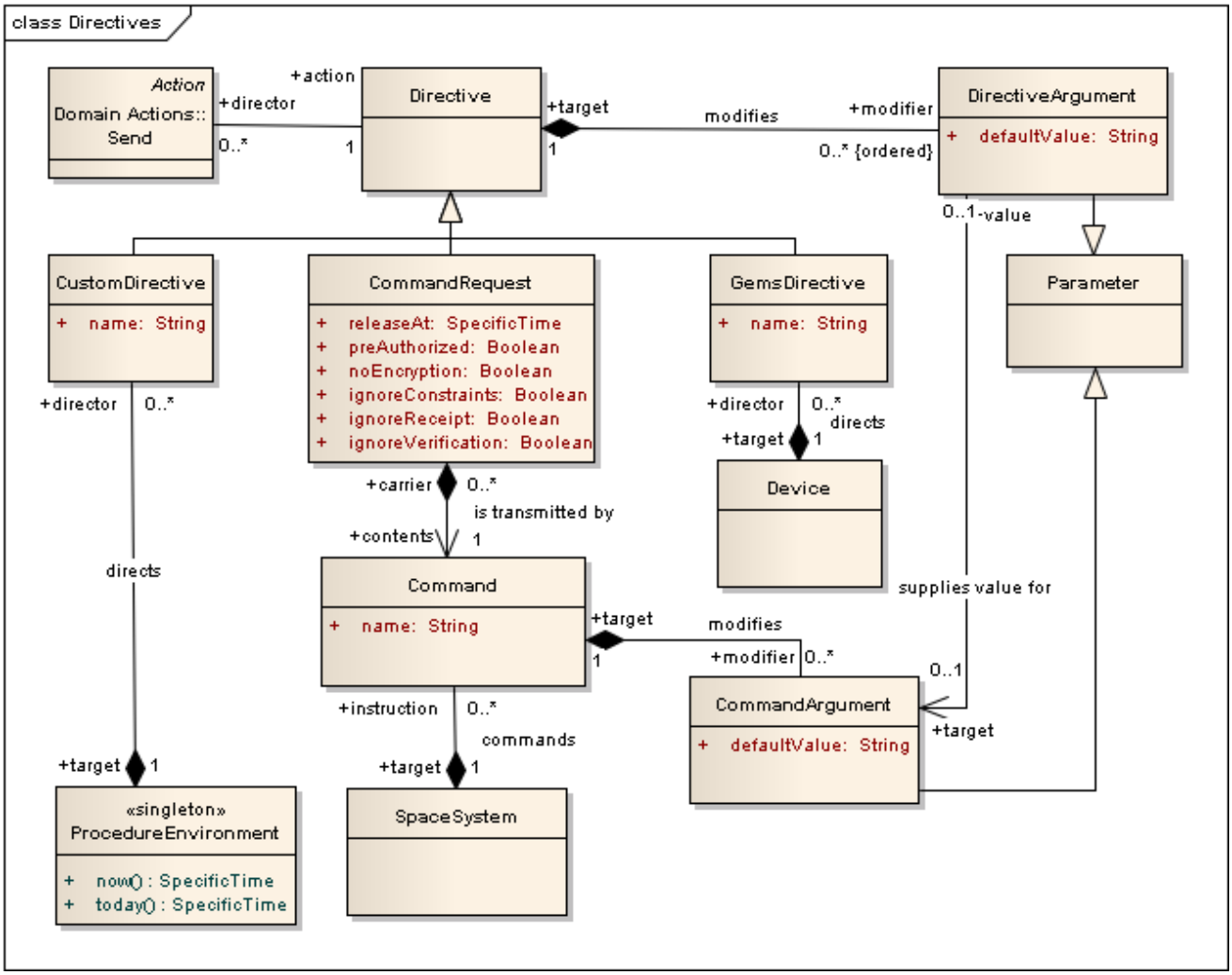


**Figure 13 Directives: CommandRequests and GemsDirectives**

### 6.5.1 Command

A Command is usually a binary packet sent to a spacecraft to change the onboard configuration. In SOLM Commands are defined by an XTCE document. The binary format is not important to SOLM, it is formatted by the ground system.

#### 6.5.1.1 Generalizations

None

#### 6.5.1.2 Attributes

- name:String – the name of the Command, unique for a specific SpaceSystem

#### 6.5.1.3 Associations

- modifier:CommandArgument[0..*] – an argument modifies a command, usually altering the contents of the binary packet sent to the spacecraft.
- carrier:CommandRequest[0..*] – a CommandRequest is part of a procedure that carries a command to the ground system for transmission.
- target:SpaceSystem[1] – a Command is intended for one target SpaceSystem.

#### 6.5.1.4 Constraints

No additional constraints

#### 6.5.1.5 Semantics

Much of the Command structure defined in an XTCE document is irrelevant for SOLM. The Procedure determines the sequence, timing, and argument values for each Command, so the only Telecommand elements SOLM needs from the XTCE document are a list of valid commands and the type and range of each command argument.

### 6.5.2 CommandArgument

A CommandArgument modifies the effect of a Command on the target SpaceSystem.

#### 6.5.2.1 Generalizations

- Parameter, page 32

#### 6.5.2.2 Attributes

- defaultValue:String – provides a default value for the argument, if none is supplied by the Directive

#### 6.5.2.3 Associations

- target:Command[1] – the Command modified by this argument
- value:DirectiveArgument[0..1] – the Directive may supply a DirectiveArgument to be used instead of the defaultValue attribute.

#### 6.5.2.4 Constraints

No additional constraints

#### 6.5.2.5 Semantics

A CommandArgument that does not have a defaultValue, must have a value supplied by a directiveArgument.

### 6.5.3 CommandRequest

A CommandRequest includes additional information about how the Command should be transmitted by the ground system.

#### 6.5.3.1 Generalizations

- Directive, page 40

#### 6.5.3.2 Attributes

- ignoreConstraints:Boolean – if true, the ground system must ignore any pre-transmission constraints defined for the command and allow the transmission to proceed without signalling an error, even if transmission would violate the constraints. Pre-transmission constraints may be defined in XTCE but are not managed by SOLM.
- ignoreReceipt:Boolean – if true, the ground system must ignore any receipt acknowledgement normally required for the command and proceed without signalling an error.
- ignoreVerification:Boolean – if true, the ground system must ignore any functional verification defined for the command, and proceed without signalling an error, even if the function verification would fail.
- noEncryption:Boolean – if true, the ground system must transmit the command without encrypting the binary packet.
- preAuthorized:Boolean – if true, the ground system must ignore any requirement for release authorization before transmission.
- releaseAt:AbsoluteTime – if provided, the ground system must not release the command for transmission until the specified time.

#### 6.5.3.3 Associations

- contents:Command[1] – the command to request the ground system to transmit.

#### 6.5.3.4 Constraints

No additional constraints

#### 6.5.3.5 Semantics

When the SOLM execution environment provides the CommandRequest directive information to the ground system, the ground system must format the command and transmit it to the SpaceSystem, conducting any normal verifications for the command, unless overridden by one of the CommandRequest attributes. Control must be returned to the executing procedure after transmission is complete, unless the "releaseAt" time is specified, in which case control must be returned after the Command is queued for transmission at a later time.

### 6.5.4 CustomDirective

A CustomDirective is a ground system-specific directive. Many scripting languages have directives that are not related to a GEMS device or a SpaceSystem. SOLM provides a way to capture CustomDirective information in a Procedure, but the system-specific behaviour is not directly transferable.

#### 6.5.4.1 Generalizations

- Directive, page 40.

#### 6.5.4.2 Attributes

name:String – the name of the directive

#### 6.5.4.3 Associations

- target:ProcedureEnvironment[1] – the environment of the CustomDirective

#### 6.5.4.4 Constraints

No additional constraints

#### 6.5.4.5 Semantics

The complete behaviour of a CustomDirective is indeterminate. For the purposes of SOLM, the name and DirectiveArguments are supplied to the ground system. The ground system either completes the directive without error, returning control to the executing Procedure, or the ground system returns an error, which will either terminate execution or be handled by an exception handler.

### 6.5.5 Directive

A Directive instructs the ground system to take an action during Procedure execution.

#### 6.5.5.1 Generalizations

None

#### 6.5.5.2 Attributes

No additional attributes.

#### 6.5.5.3 Associations

modifier:DirectiveArgument[0..*] – supplies additional information to the ground system for completing the action of the Directive.

#### 6.5.5.4 Constraints

No additional constraints

#### 6.5.5.5 Semantics

A Directive is an abstract representation of a ground system action. See the related concrete Directive descendants for semantics.

### 6.5.6 DirectiveArgument

A DirectiveArgument supplies additional information to the ground system for Directive execution.

#### 6.5.6.1 Generalizations

- Parameter, page 32

#### 6.5.6.2 Attributes

- defaultValue: String – provides a default for the argument value

#### 6.5.6.3 Associations

- target:CommandArgument[0..1] – CommandArgument that will receive the DirectiveArgument value.

#### 6.5.6.4 Constraints

No additional constraints

#### 6.5.6.5 Semantics

DirectiveArguments are defined during procedure definition and are passed to the ground system during Directive execution by the Send action. See page 44.

### 6.5.7 GemsDirective

A GemsDirective sends a command and parameters to a GEMS device.

#### 6.5.7.1 Generalizations

- Directive, page 40

#### 6.5.7.2 Attributes

- name:String – the directive name

#### 6.5.7.3 Associations

- target:Device – the GEMS device targeted by this GemsDirective

#### 6.5.7.4 Constraints

No additional constraints

#### 6.5.7.5 Semantics

The execution of a GemsDirective issues a directive message to the GEMS device. The GemsDirective is completed and control returns to the Procedure execution when the GEMS device responds. If the GEMS device fails to respond or returns an error response, the GemsDirective will return an error, halting procedure execution or transferring control to an exception handler, if one is defined for the procedure.

## 6.6 Procedure Actions

Spacecraft operations procedures frequently check the current value of a telemetry parameter to determine if a command was properly executed or to determine the correct command to send, based on the current spacecraft state. Requested parameters can be used in conditional expressions or as command arguments within the procedure. In SOLM, the telemetry Parameter instances are created from the XTCE document in the modeling environment. The value of the Parameter instance may be referenced in the procedure model for conditional tests, computation, or setting the value of a Directive Parameter.

Figure 14 through Figure 21 show the action nodes that can be part of an activity diagram defining an operations procedure.

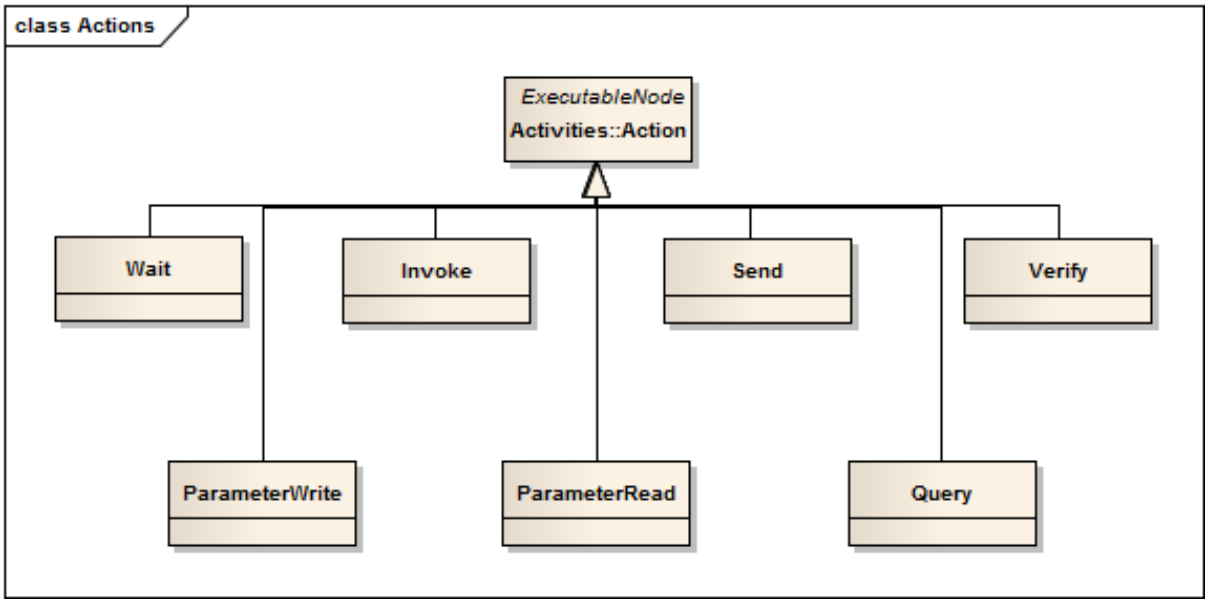


**Figure 14 SOLM Action Nodes for Activity Diagrams**

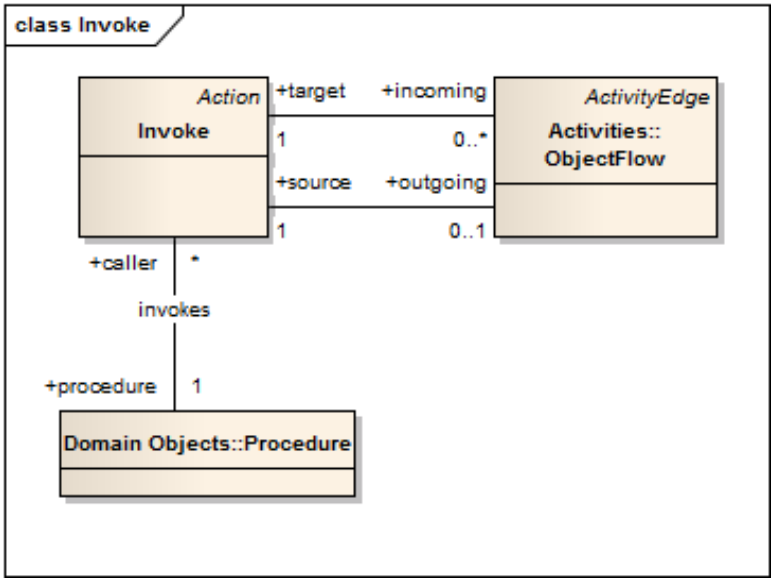


**Figure 15 Invoke Subprocedure**

### 6.6.1 Invoke

A Procedure may invoke another Procedure as a sub-procedure.

#### 6.6.1.1 Generalizations

- Action, page 17

#### 6.6.1.2 Attributes

No additional attributes.

#### 6.6.1.3 Associations

No additional associations.

#### 6.6.1.4 Constraints

No additional constraints

#### 6.6.1.5 Semantics

The Invoke action calls a Procedure as a sub-procedure. Any ProcedureArguments required by the sub-procedure definition must be supplied from the associated incoming ObjectFlows. Execution of ModeledProcedures take place in the same ProcedureEnvironment context, but there may be InternalParameters that are local to the ModeledProcedure. The Integer result of the Procedure may be passed to an outgoing ObjectFlow. If the sub-procedure completes normally, the control token is passed to the outgoing ControlFlow of the Action. If the sub-procedure exits with an error, control is passed to error handling within the current Activity.

### 6.6.2 ParameterRead

A ParameterRead is required to obtain a new value for an ExternalParameter.

#### 6.6.2.1 Generalizations

- Action, page 15

#### 6.6.2.2 Attributes

No additional attributes.

#### 6.6.2.3 Associations

- incoming:ObjectFlow[1] – ObjectFlow connected to an ExternalParameter providing a new value.
- outgoing:ObjectFlow[1] – ObjectFlow connected to a ProcedureParameter receiving the new value.

#### 6.6.2.4 Constraints

No additional constraints

#### 6.6.2.5 Semantics

An ExternalParameter requires an interaction with the ground system to obtain a new value for the Parameter.

### 6.6.3 ParameterWrite

A ParameterWrite sets the value of a Parameter.

#### 6.6.3.1 Generalizations

- Action, page 15

#### 6.6.3.2 Attributes

No additional attributes.

#### 6.6.3.3 Associations

- value:Expression[1] – Expression evaluated to obtain the new value for the Parameter
- outgoing:ObjectFlow[1] – ObjectFlow connected to a Parameter receiving the new value.

#### 6.6.3.4 Constraints

No additional constraints

#### 6.6.3.5 Semantics

An ExternalParameter requires an interaction with the ground system to set a new value for the Parameter.

### 6.6.4 Query

A Query obtains a new value for a Parameter from the Operator.

#### 6.6.4.1 Generalizations

- Action, page 15

#### 6.6.4.2 Attributes

- prompt:String – text to prompt the operator for a value.

#### 6.6.4.3 Associations

- outgoing:ObjectFlow[1] – ObjectFlow connected to the Parameter receiving the new value.

#### 6.6.4.4 Constraints

No additional constraints

#### 6.6.4.5 Semantics

There are system-specific ways to prompt the operator for a value. The ParameterType and prompt can be provided to insure a good value is provided. Failure to provide a value or providing a value that is not allowed by the ParameterType and Restrictions results in an error.

### 6.6.5 Send

A Send issues a Directive to the ground system for execution.

#### 6.6.5.1 Generalizations

- Action, page 15

#### 6.6.5.2 Attributes

No additional attributes.

#### 6.6.5.3 Associations

- action:Directive[1] – the Directive to send to the ground system.
- incoming:ObjectFlow[0..*] – ObjectFlows providing values from InternalParameters
- outgoing:ObjectFlow[0..*] – ObjectFlows providing values to DirectiveArguments

#### 6.6.5.4 Constraints

No additional constraints

#### 6.6.5.5 Semantics

A Send action collects DirectiveArgument values and issues the Directive to the ground system for completion.

### 6.6.6 Verify

Verify is an abstract action.

#### 6.6.6.1 Generalizations

- Action, page 15

#### 6.6.6.2 Attributes

No additional attributes.

#### 6.6.6.3 Associations

No additional associations.

#### 6.6.6.4 Constraints

No additional constraints

#### 6.6.6.5 Semantics

See VerifyExpression and VerifyRange

### 6.6.7 VerifyExpression

VerifyExpression evaluates a Boolean expression.

#### 6.6.7.1 Generalizations

- Verify, page 44

#### 6.6.7.2 Attributes

- expression:String – string containing a Python Boolean expression.

#### 6.6.7.3 Associations

No additional associations.

#### 6.6.7.4 Constraints

The expression must be a valid Python Boolean expression. The syntax for Python is maintained in an open-source project at https://docs.python.org/reference.

#### 6.6.7.5 Semantics

The expression is evaluated and if it is true, execution continues at the outgoing ControlFlow. If it is false, an error is generated and any associated exception handler is executed.

### 6.6.8 VerifyRange

A VerifyRange tests the equality of a floating point Parameter type.

#### 6.6.8.1 Generalizations

- Verify, page 44

#### 6.6.8.2 Attributes

- expected:String – the expected value of the Parameter.
- tolerance:String – a plus/minus tolerance value to test for equality within a range.

#### 6.6.8.3 Associations

- readReference:InternalParameter[1] – the Parameter to test for equality

#### 6.6.8.4 Constraints

No additional constraints

#### 6.6.8.5 Semantics

The value of the readReference is compared to the range defined by the value of the expected attribute of the VerifyRange, plus the tolerance and minus the tolerance. If it is within the range, inclusively, execution continues at the outgoing ControlFlow. If it is not, an error is generated and any associated exception handler is executed.

### 6.6.9 Wait

Wait is an abstract action.

#### 6.6.9.1 Generalizations

- Action, page 15

#### 6.6.9.2 Attributes

No additional attributes.

#### 6.6.9.3 Associations

No additional associations.

#### 6.6.9.4 Constraints

No additional constraints

#### 6.6.9.5 Semantics

See WaitOnExpression and WaitOnTime for specific semantics.

### 6.6.10 WaitOnExpression

WaitOnExpression waits for the value of a Boolean expression to become true.

#### 6.6.10.1 Generalizations

- Wait, page 45

#### 6.6.10.2 Attributes

- pollPeriod:TimeInterval – the time period to wait before re-evaluating the expression.
- timeout:Time – the time period or AbsoluteTime to wait before failing due to timeout

#### 6.6.10.3 Associations

- expression:Expression[1] – the Boolean expression to evaluate

#### 6.6.10.4 Constraints

No additional constraints

#### 6.6.10.5 Semantics

This Action repeatedly evaluates the expression, obtaining new values for ExternalParameters referenced, until the Expression is true or the timeout occurs. Execution continues on the outgoing ControlFlow, if the expression is true. The timeout error will transfer control to an exception handler, if one is defined for the Action.

When a TimeInterval is used as the timeout in a WaitOnExpression action, it is used as the time interval to wait for the expression to become true. A zero or negative interval will result in an immediate timeout error if the expression is not true. When used as the pollPeriod in a WaitOnExpression action, it is used as the time interval to wait between expression evaluations. A zero or negative interval in this case will cause an immediate error.

### 6.6.11 WaitOnTime

WaitOnTime delays the execution thread.

#### 6.6.11.1 Generalizations

- Wait, page 45

#### 6.6.11.2 Attributes

- time:Time – the TimeInterval period to wait, or the AbsoluteTime to resume execution.

#### 6.6.11.3 Associations

No additional associations.

#### 6.6.11.4 Constraints

No additional constraints

#### 6.6.11.5 Semantics

Execution suspends until the specified time is reached, then execution continues on the outgoing ControlFlow. When a TimeInterval is used as the time in a WaitOnTime action, it defines the interval to delay before continuing execution. A zero or negative interval will result in an immediate completion of the action. When a SpecificTime is used as the time in a WaitOnTime action, execution will delay until

the SpecificTime is reached. If the SpecificTime is in the past, it results in an immediate completion of the action.

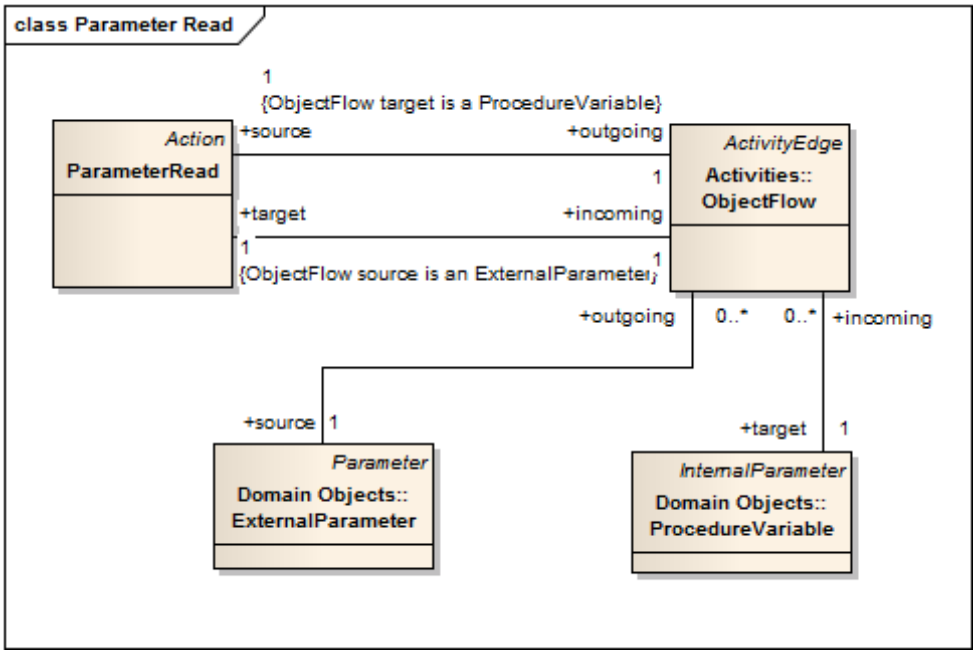


**Figure 16 Parameter Read**

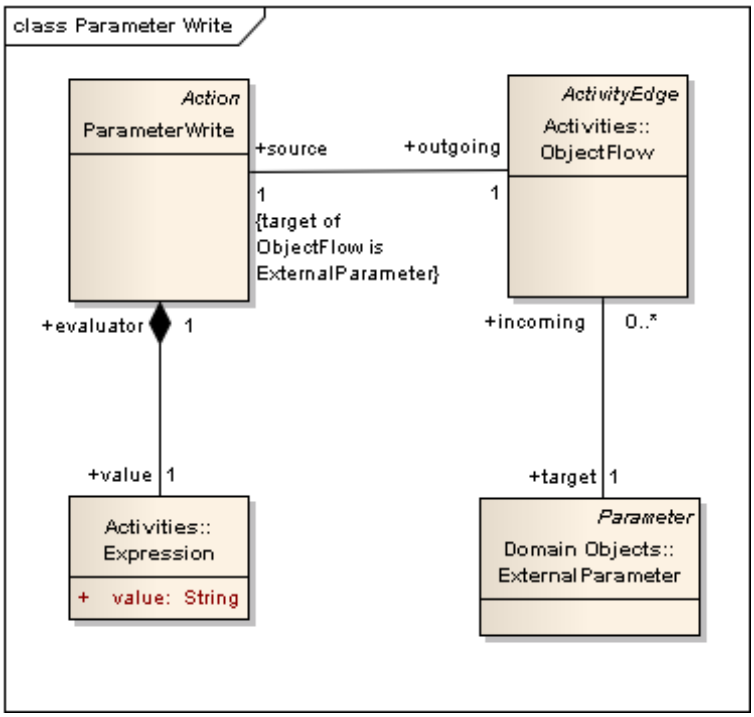


**Figure 17 Parameter Write**

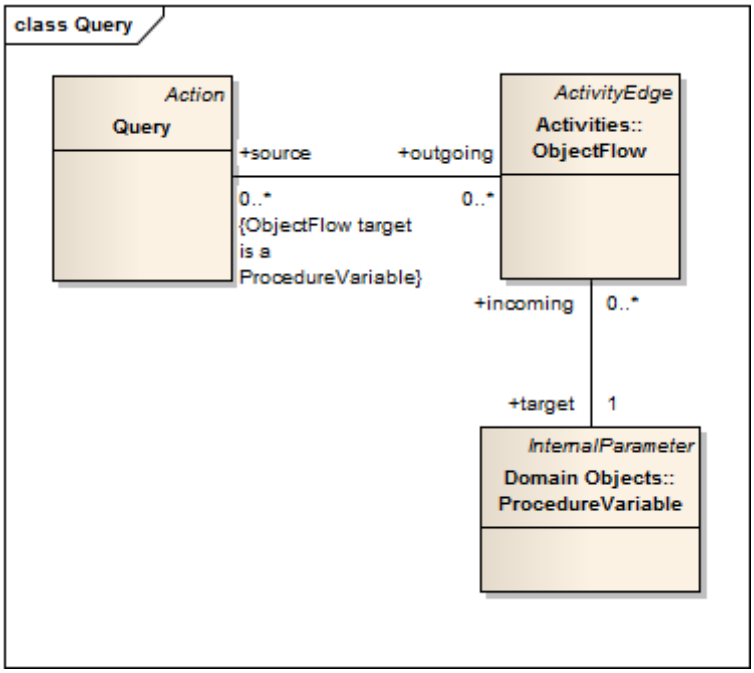


**Figure 18 Query Operator**

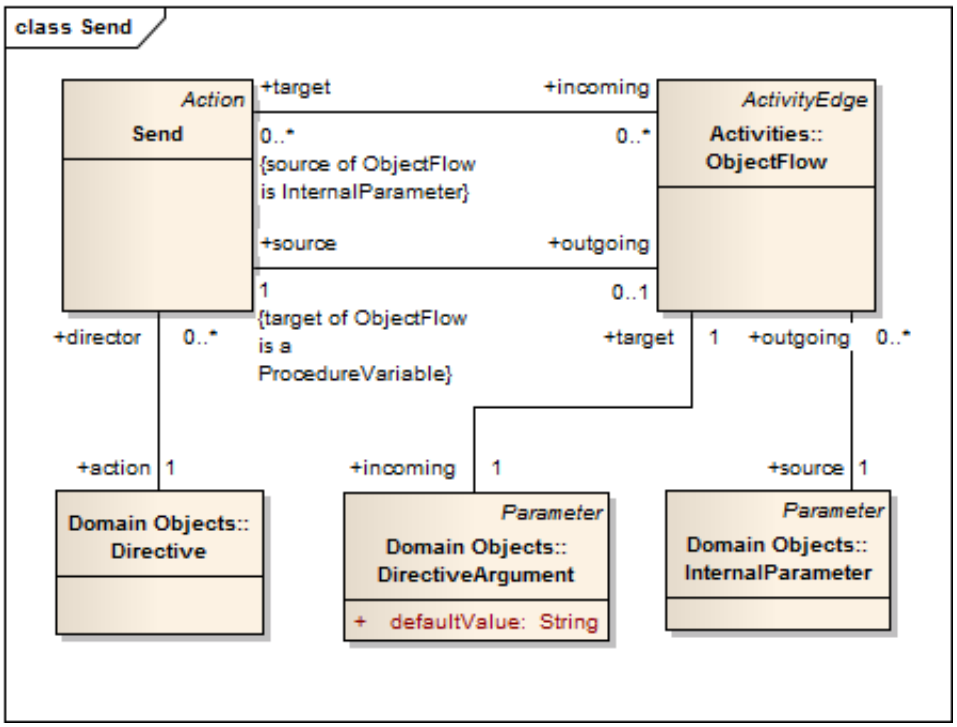


**Figure 19 Send Directive**

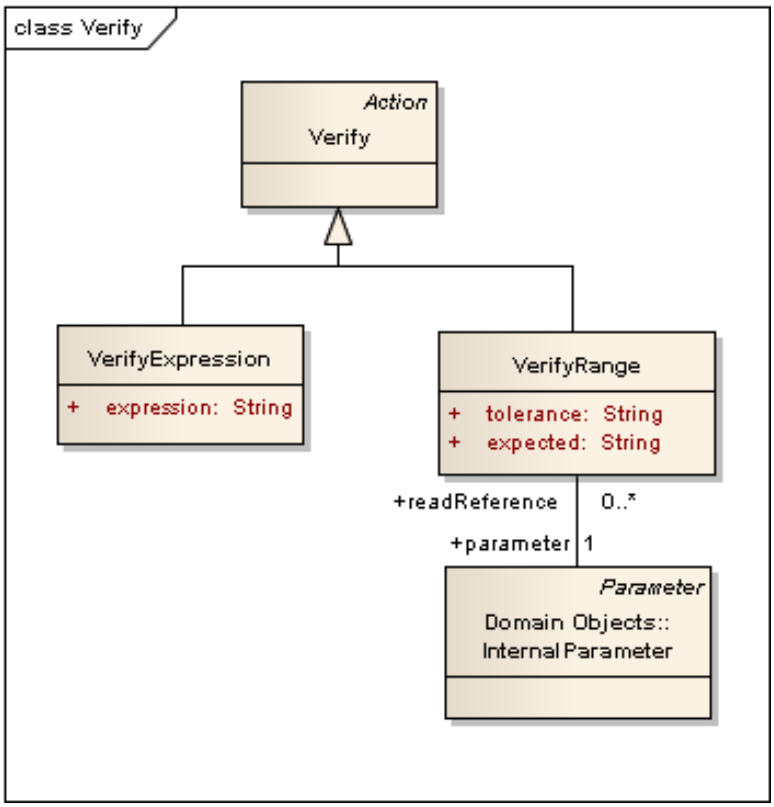


**Figure 20 Verify State**

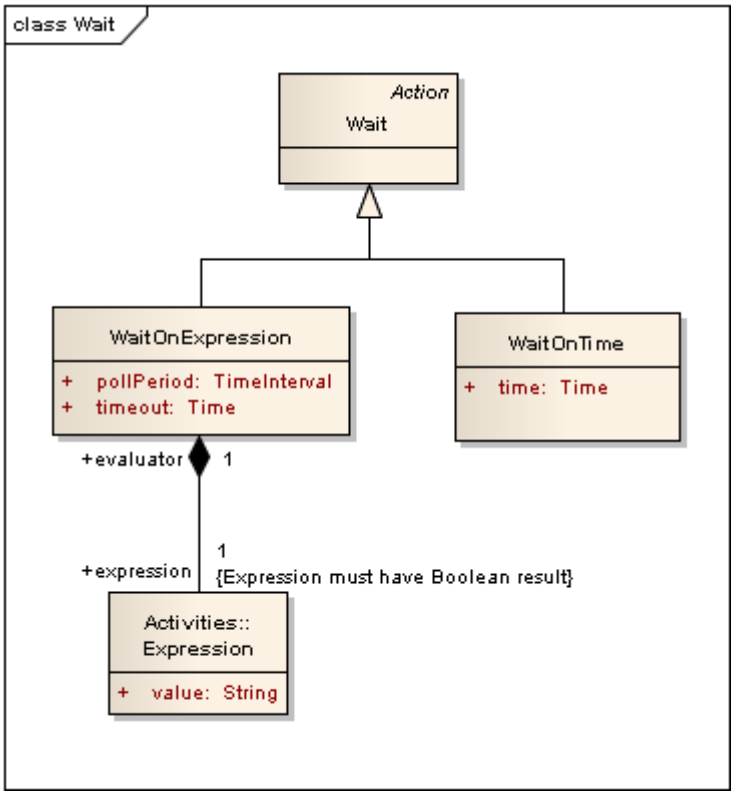


**Figure 21 Wait**

# 7 SpacePython Mapping

## 7.1 General

Python is an open-source scripting language for general purpose computing applications that can easily be extended to specialized applications, such as spacecraft control. This section provides information on mapping SOLM to a SpacePython script. SpacePython is based on the Python 3 syntax with the addition of the SOLM extensions defined in this appendix to support spacecraft operations.

Because either direct execution or translation of a SpacePython procedure is an acceptable conformance to SOLM, additional machine consumable SpacePython script examples are provided with the SOLM specification, including the SetMomentumWheelSpeed example from Figure 4. All of the examples are provided in the form of Python package that can be added to any Python installation. The example module will run all of the example scripts, demonstrating a compliant SpacePython interface. A SOLM-compliant ground system would use its own database and control mechanisms to run the same SpacePython scripts rather than modules provided in the demonstration space package.

| SpacePython | SOLM | Comments |
|---|---|---|
| # | Comment | Delimits comment text |
| While | Loop defined by DecisionNode, ControlFlow | Defines a loop construct with a conditional entrance. The Python language also supports a for construct with iteration over lists. Indentation is significant for defining the span of the loop.<br>This is in the Python base language. |
| If..[elif]..[else] | DecisionNode | Defines conditional execution paths. Indentation is significant for defining the span of each conditional block.<br>This is in the Python base language. |
| Invoke | InitialNode | Defines the primary entry point for a procedure accepting keyword arguments for the parameter values.<br>This is SpacePython usage of a function definition as the procedure entry point. |

| | | |
|---|---|---|
| operatorQuery | Query | Python supports numerous GUI widget sets and console input, but a SpacePython procedure uses an operatorQuery function that interfaces with the execution platform GUI<br>def operatorQuery(prompt, parameterList):<br>This is a SpacePython library function |
| operatorQuery(‘prompt text’, None) | Query with no associated Parameters | An operatorQuery with no parameters is essentially a PAUSE, waiting for the operator to continue the procedure execution.<br>This is a SpacePython library function |
| Asset | SpaceSystem | An Asset object represents a device including one that provides command and telemetry connection through the ground system to a specific SpaceSystem. If the underlying system needs additional information or settings to establish the Asset, these may be supplied in a system-specific way using the SpaceSystem name as a key.<br>This is a SpacePython class definition. |
| Asset.send(<COMMAND>) | CommandRequest<br>Command | The send method of an Asset object issues a command transmission request to the ground system for the SpaceSystem associated with the Asset. A fully-specified command (no required Arguments) can be specified by its unique name, otherwise a Command instance must be obtained from the Asset.lookupCommand() method |

| | | |
|---|---|---|
| | | and completed by specifying required arguments to the Command instance.<br>This is a SpacePython class method. |
| Asset.lookupParameter(<PARAMETER>) | XtceParameter<br>ParameterType<br>Restriction | Obtain an instance of a Parameter in the SpaceSystem associated with the Asset.<br>This is a SpacePython class method. |
| Asset.lookupCommand(<COMMAND>) | Command | Obtain the current values of a list of Parameters, specified by unique names, from a device.<br>This is a SpacePython class method. |
| Asset (<DEVICE>) | Device | An Asset object represents a connection to a specific device. The object is created with the unique name for the device, which may be mapped via a system-specific configuration to the network address of the device. This is a SpacePython class definition. |
| Asset.get() | GemsParameter<br>ParameterRead | Obtain the current values of a list of Parameters, specified by unique names, from a device.<br>This is a SpacePython class method. |
| Asset.setParameters() | GemsParameter<br>ParameterWrite | Set the values of a list of Parameters in a device, specified as a list of name=value pairs.<br>This is a SpacePython class method. |
| Asset.lookupCommand() | GemsDirective<br>Restriction | Obtain an instance of a Command, specified by unique name, defined for the device.<br>This is a SpacePython class |

| | | |
|---|---|---|
| | | method. |
| Asset.send() | GemsDirective | Issues a Command to the device. The Directive instance must be obtained by the Asset.lookupCommand() method and completed by supplying any required Parameter values.<br>This is a SpacePython class method. |
| <Parameter> | Parameter | Represents a Parameter.<br>This is a SpacePython abstract class. |
| <Parameter>__<Attribute> | Restriction | Provides the attributes of a parameter including limits and valid range. |
| Parameter.value() | XtceParameter<br>ParameterRead | Returns the engineering unit value of the Parameter for use in an expression.<br>This is a SpacePython class method. |
| Parameter.raw() | XtceParameter | Returns the raw (usually binary or integer) value of the Parameter for use in an expression.<br>This is a SpacePython class method. |
| try … catch | HandledExceptionRegion | The try catch block sets up an exception handler for a protected section of the procedure. |
| verify | Verify | A statement that accepts a Boolean expression and raises an exception if the condition is false. |
| Parameter | GemsParameter | Represents a Parameter definition. Must be obtained by a factory method “lookupParameter” on the Asset.<br>This is a SpacePython class. |

| | | |
|---|---|---|
| Parameter | XtceParameter | Represents an Parameter definition. Must be obtained by a factory method "lookupParameter" on Asset. |
| wait(seconds) | WaitOnTime | Wait for a time interval in seconds. |
| waitUntil(<SpecificTime>) | WaitOnTime | Wait for a specific date/time. |
| waitFor(<expression>,<timeout>,<Polling period>) | WaitOnExpression | Wait for an expression to become true or a timeout occurs. |

| SOLM | SpacePython | Comments |
|---|---|---|
| InitialNode | `from space import Asset`<br>`def invoke(**kwargs):` | Python accepts param=value style keyword arguments. |
| SpaceSystem | asset = Asset("<Device.SpaceSystem>") | An Asset must be defined in the procedure class model in order to access Parameters and Commands |
| Device | dev = Asset("<Device.name>") | An Asset must be defined in the procedure class model in order to access Parameters and Commands |
| DecisionNode | if <guard1 expression>:<br>…<br>elif<guard2 expression>:<br>...<br>else <default guard>:<br>... | The guard paths should be exclusive, since there is no order implied in the activity diagram other than the default guard. |
| JoinNode | Thread.join() | De-indentation completes a conditional or loop. Join() waits for the joined thread |

| | | |
|---|---|---|
| | | to terminate. |
| ParameterRead | Value = dev.lookupParameter('<Parameter.name>').value() | |
| ParameterWrite | dev.setParameters(<Parameter.name>=value) | |
| Send | dev.send(directive) | The Send action is translated based on the Command type. |
| ParameterWrite | dev.setParameters(<Parameter.name>=value) | |
| ActivityFinalNode | return | |
| HandledExceptionRegion | try:<br>…<br>catch:<br>… | |
| WaitOnTime | wait(<seconds>)<br>waitUntil(<SpecificTime) | SpacePython allows waiting for a specific time of day or for a time interval. The time interval may include fractional seconds. |
| WaitOnExpression | waitFor(<Expression>, <Timeout>, <Polling Period>) | SpacePython allows waiting for an expression to become true or timeout period to elapse. |

## 7.2 Upgrading Notes

Changes in SpacePython 1.2 were made to address several changes in the Python language as well as the space industry since when SpacePython was originally created. SpacePython 1.2 unifies space and ground assets, which represents a change in how ground systems are organized. While the SOLM platform independent model still separately represents space and ground interfaces, the platform-specific model for SpacePython now utilizes a single construct of Assets to represent both of them. In addition, the construct of a Downlink have been removed as this is simply a space asset connection. Rather than having different functions and behaviors between space and ground assets, SpacePython 1.2 allows the same methods and functionality to be applied to both.

With SpacePython 1.2, the SpacePython library has been updated to have the abstract interface separated from concrete implementations. This allows different implementations to be developed by implementing the

abstract base classes and could be used simultaneously. It also provides improved procedure development by simplifying the methods that are made public by the SpacePython library. Additionally, this approach makes future upgrades to the specification easier for implementers, since the reference implementation has been separated.

Also, SpacePython has added type hints and documentation to all functions. This allows procedure developers to be able to significantly improve the quality of procedures being developed to use the correct data types at development time and likely reduce errors found at run-time. This levers the software development recommendation to ‘fail fast’. As such, implementers of this specification should add Python type hints to all function arguments and return values. In addition, implementers should handle all function return value types properly to account for all possible code paths that may take place.

Changes of consideration:

- The Asset.send() function takes three arguments (whereby Link.send() only took two arguments). Please update uses of the send() function to accommodate the additional argument.
- The Asset.updateParameters() function is renamed from Link.get. This reflects a better description of the function’s purpose. Please note that the use of this function is optional.
- Accessing assets is performed via the lookupAsset function on SpacePython object rather than directly calling Link(). This is because Assets are a pluggable interface.
- Accessing a procedure is now performed via procedureEngine().loadProcedure rather than calling the loadProcedure global function. This is because the procedure engine is now a plugable interface.

It is also recommended to review the sample procedures within the ‘test’ directory of the SpacePython machine readable files to review how to utilize the SpacePython classes.