Spacecraft Operations Language Metamodel (SOLM)

FTF Beta 1

OMG Document Number: dtc/2011-05-06
Standard document URL: http://www.omg.org/spec/SOLM/1.0/
Associated Schema File(s)*:   http://www.omg.org/spec/SOLM/20110301

* original file: space/2011-03-02 (XMI)

This OMG document replaces the submission document (space/2011-03-01, Alpha). It is an OMG Adopted Beta Specification and is currently in the finalization phase. Comments on the content of this document are welcome, and should be directed to *issues@omg.org* by December 18, 2011.

You may view the pending issues for this specification from the OMG revision issues web page *http://www.omg.org/issues/*.

The FTF Recommendation and Report for this specification will be published on February 19, 2012. If you are reading this after that date, please download the available specification from the OMG Specifications Catalog.

OMG's Issue Reporting Procedure

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, Report a Bug/Issue (http://www.omg.org/technology/agreement.htm).

Table of Contents

Preface

About the Object Management Group

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at *http://www.omg.org/*.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. A catalog of all OMG Specifications is available from the OMG website at:

http://www.omg.org/technology/documents/spec_catalog.htm

Specifications within the Catalog are organized by the following categories:

OMG Modeling Specifications

- UML
- MOF
- XMI
- CWM
- Profile specifications.

OMG Middleware Specifications

- CORBA/IIOP
- IDL/Language Mappings
- Specialized CORBA specifications
- CORBA Component Model (CCM).

Platform Specific Model and Interface Specifications

- CORBAservices

- CORBAfacilities
- OMG Domain specifications
- OMG Embedded Intelligence specifications
- OMG Security specifications.

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. (as of January 16, 2006) at:

OMG Headquarters
140 Kendrick Street
Building A, Suite 300
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult *http://www.iso.org*

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier - 10 pt. Bold:` Programming language elements.

Helvetica/Arial - 10 pt: Exceptions

Note – Terms that appear in *italics* are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to *http://www.omg.org/technology/agreement.htm*.

1   Scope

1.1   General

This specification defines a meta-model, Spacecraft 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.

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.
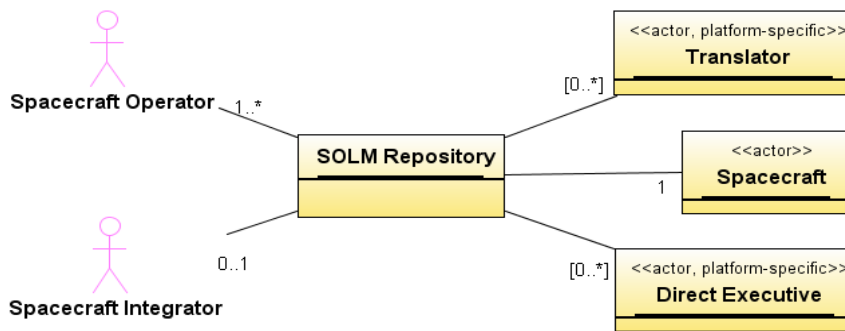
Figure 1 - SOLM Context

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.
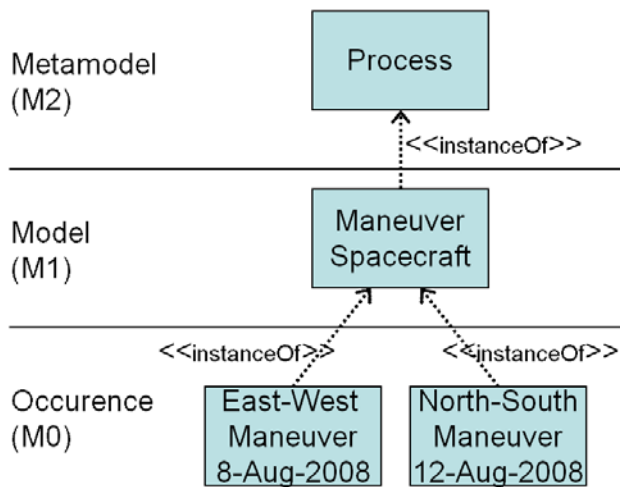
Figure 2 - Process Modeling

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 DSLs may be targeted by a translator.

A SOLM Modeling Tool maintains a repository of spacecraft procedure PIMs and can export PSMs 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 models 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 that 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 (ptc/10-11-16, ptc/10-11-14)

MOF 2.4 Core Specification (ptc/10-09-05)

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 (ptc/10-09-10)

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 1.1, (formal/2010-10-02)

4   Terms and Definitions

For the purposes of this specification, the following terms and definitions 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 - Spacecraft 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 Metamodel 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.
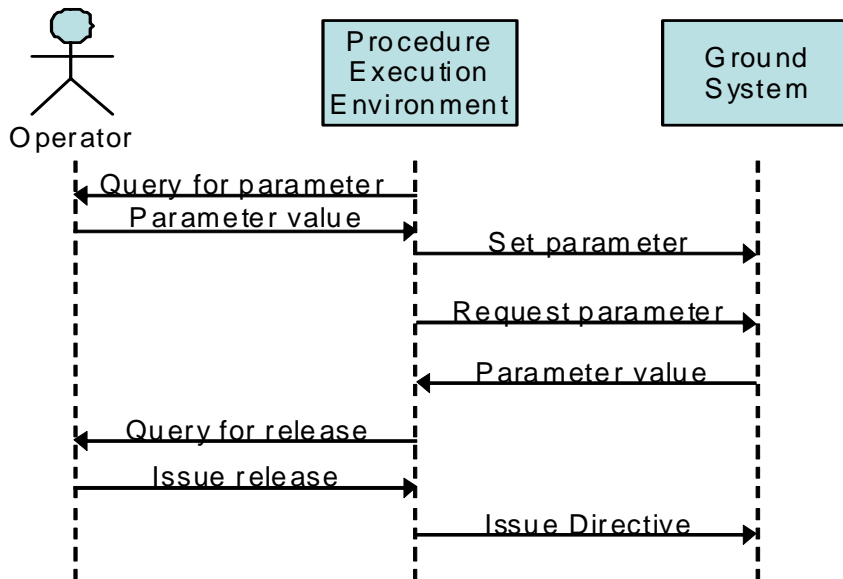
Figure 3 - Notional System Sequence Diagram (Non-Normative)

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 specification 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, and the Comet Control Language (CCL), mapped to SOLM in this specification, supports early/error termination.
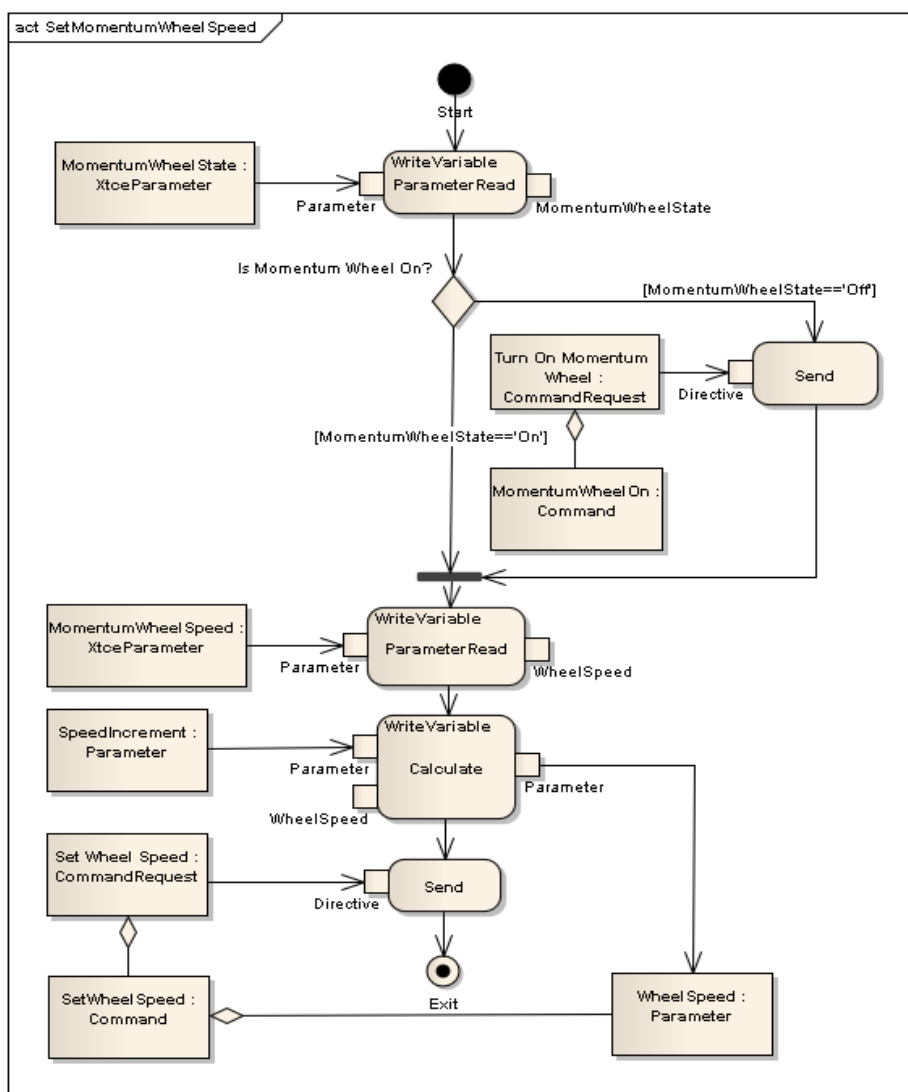
Figure 4 - Notional SOLM Procedure Activity Diagram (Non-Normative)

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.
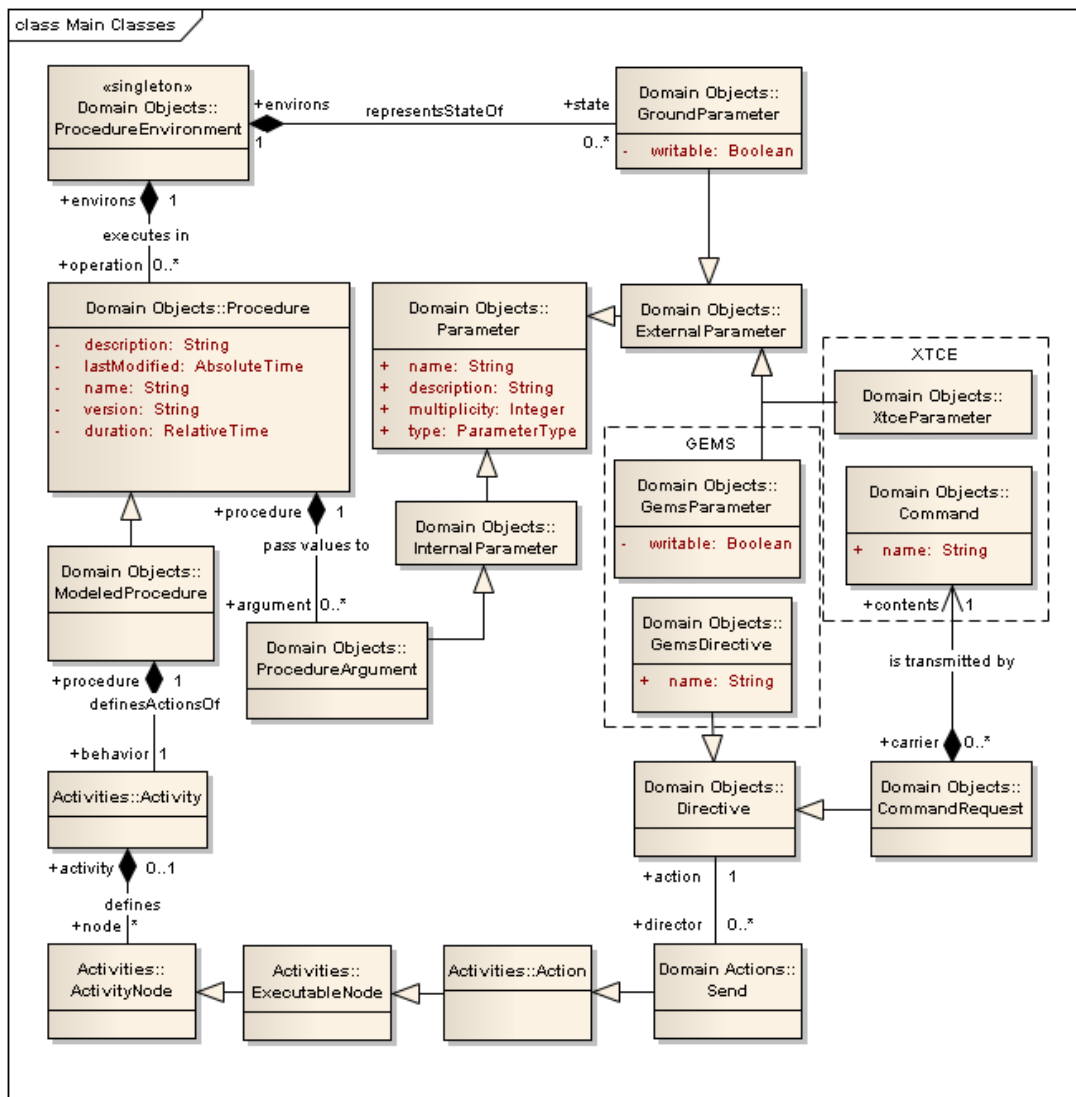
Figure 5 - Key SOLM Classes

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.
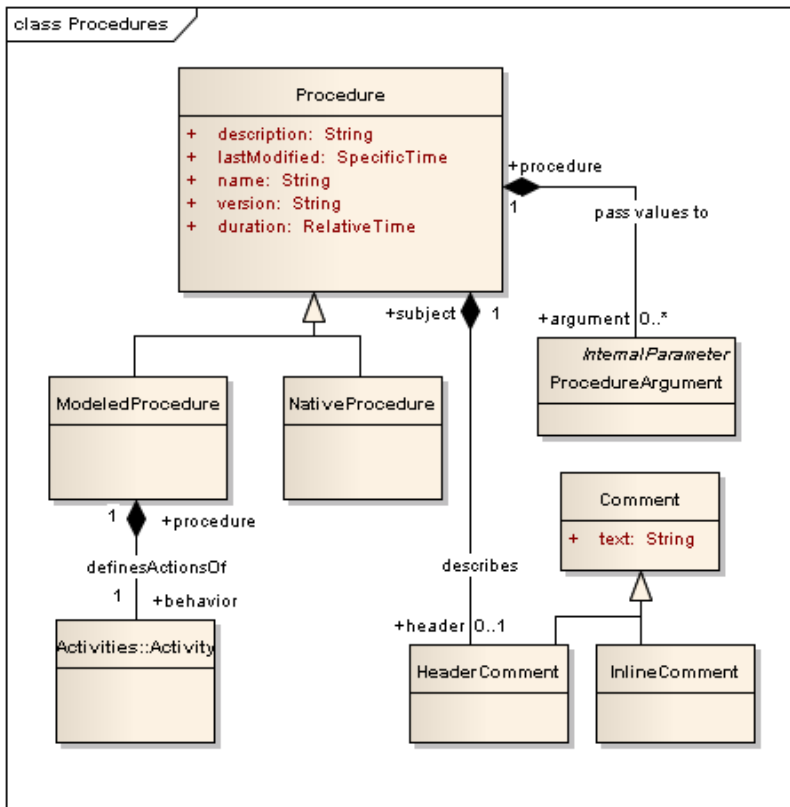
Figure 6 - Procedure Signatures

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 Modeled Procedure

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

Generalizations

- "Procedure" on page 10

Attributes

No additional attributes.

Associations

- behavior:Activity - the modeled behavior of the Procedure.

Constraints

No additional constraints.

Semantics

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

6.2.2 Native Procedure

A NativeProcedure is a Procedure that can be invoked by another Procedure in SOLM, but does not have a modeled behavior. It is defined in the modeling environment so that it can be invoked by modeled 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 modeled in SOLM due to system-specific extensions.

Generalizations

- "Procedure" on page 10

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints.

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

Generalizations

None

Attributes

- description:String - A short text description of the effects of the procedure that can be presented to a modeler 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.

Associations

- argument:ProcedureArgument[0..*] - arguments to the procedure that can receive specific values at execution time to modify the behavior 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.

Constraints

No additional constraints.

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 Procedure Argument

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

Generalizations

- "InternalParameter" on page 30

Attributes

No additional attributes.

Associations

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

Constraints

No additional constraints.

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 behavior, but the changed value is not returned to a calling Procedure.

6.2.5   Comment

A Comment is descriptive text in a Procedure.

Generalizations

None

Attributes

- text:String - The descriptive text.

Associations

No additional associations.

Constraints

No additional constraints.

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

Generalizations

- "Comment" on page 11

Attributes

No additional attributes.

Associations

- subject:Procedure - the Procedure this comment describes.

Constraints

No additional constraints.

Semantics

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

6.2.7   InlineComment

Generalizations

- "Comment" on page 11

Attributes

No additional attributes.

Associations

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

Constraints

No additional constraints.

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.
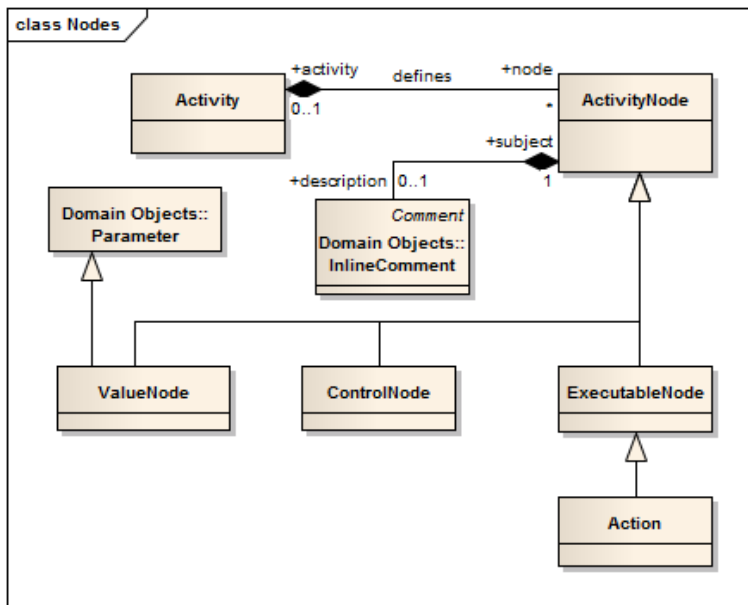
Figure 7 - Activities
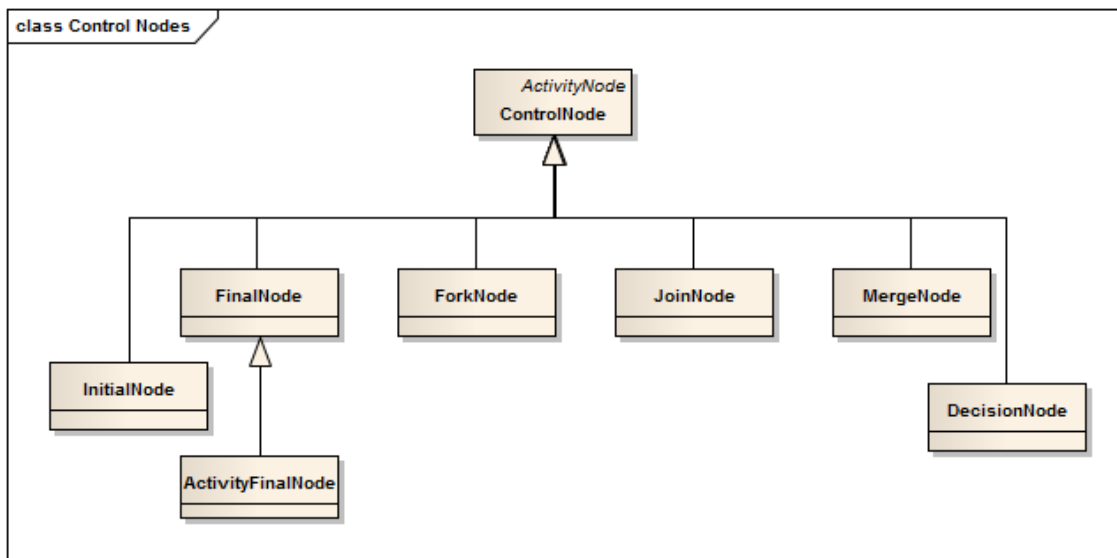
Figure 8 - Control Nodes
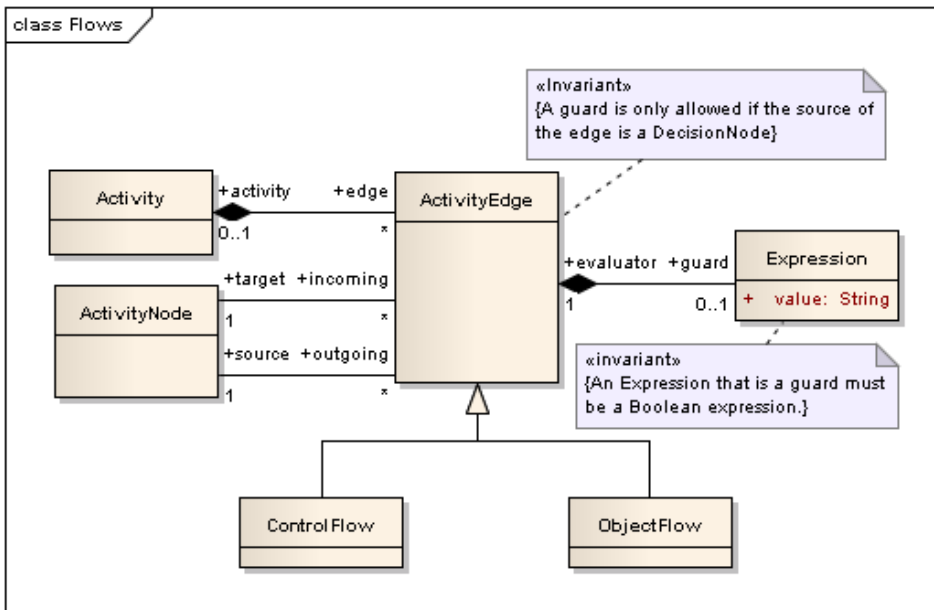
Figure 9 - Control and Data Flow
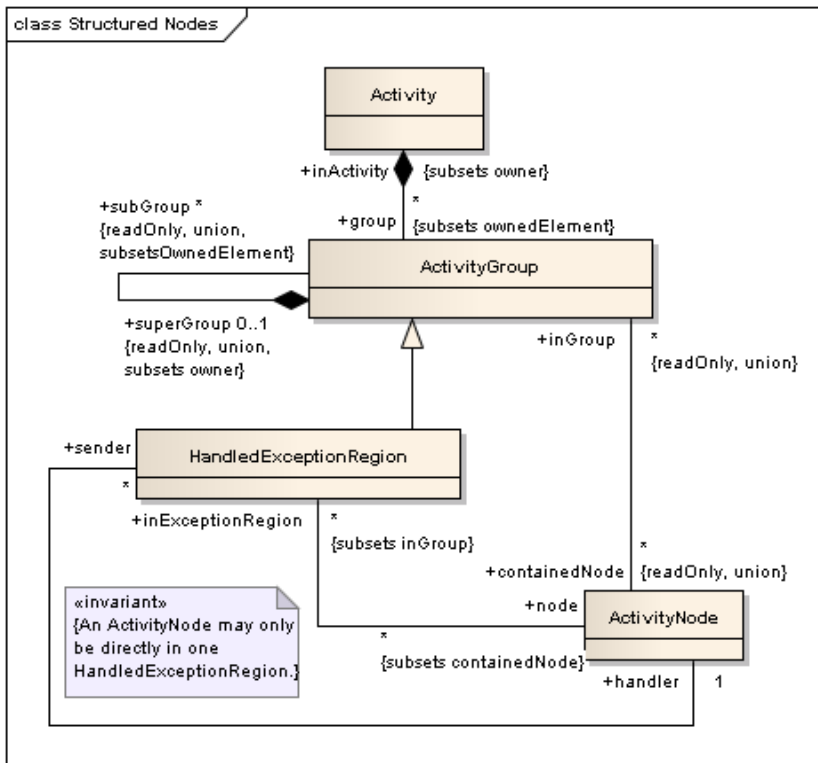
Figure 10 - Exception Handling

6.3.1 Action

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

Generalizations

- "ExecutableNode" on page 20

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints.

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 behavior of a ModeledProcedure.

Generalizations

None

Attributes

No additional attributes.

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 behavior for.

Constraints

No additional constraints.

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.

Generalizations

None

Attributes

No additional attributes.

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.

Constraints

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

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.

Generalizations

- "FinalNode" on page 21

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints

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.

Generalizations

None

Attributes

No additional attributes.

Associations

- containedNode:ActivityNode[*] - nodes in group

Constraints

No additional constraints

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.

Generalizations

None

Attributes

No additional attributes.

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[*] - outgoing 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.

Constraints

No additional constraints

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.

Generalizations

- "ActivityEdge" on page 16

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints

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).

Generalizations

- "ActivityNode" on page 18

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints

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.

Generalizations

- "ControlNode" on page 19

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

- A DecisionNode has one incoming edge and at least one outgoing edge.

- The incoming and outgoing edges must be all control flows.

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 behavior is unspecified. Only one ControlFlow will be activated by a decision node.

6.3.10 ExecutableNode

An ExecutableNode performs domain-specific actions.

Generalizations

- "ActivityNode" on page 18

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints.

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.

Generalizations

None

Attributes

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

Associations

No additional associations.

Constraints

The expression must be a valid python arithmetic expression (version 2.7). The syntax for python is maintained in an open source project at http://docs.python.org/reference.

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.

Generalizations

- "ControlNode" on page 19

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints.

Semantics

See ActivityFinalNode, page Error! Bookmark not defined.

6.3.13 ForkNode

A ForkNode starts one or more parallel threads of execution.

Generalizations

- "ControlNode" on page 19

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints.

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.

Generalizations

- "ActivityGroup" on page 17

Attributes

No additional attributes.

Associations

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

Constraints

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

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.

Generalizations

- "ControlNode" on page 19

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints.

Semantics

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

6.3.16 JoinNode

A JoinNode ends parallel threads of execution.

Generalizations

- "ControlNode" on page 19

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

- A JoinNode has exactly one outgoing edge that is a ControlFlow

- All incoming edges are ControlFlows

- There is at least one incoming edge.

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.

Generalizations

- "ControlNode" on page 19

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

- A MergeNode has exactly one outgoing edge.

- A MergeNode has at least one incoming edge.

- All incoming and outgoing edges are ControlFlows.

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.

Generalizations

- "ActivityEdge" on page 16

Attributes

No additional attributes.

Associations

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

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

Constraints

No additional constraints.

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.

Generalizations

- "ActivityEdge" on page 16

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints.

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.
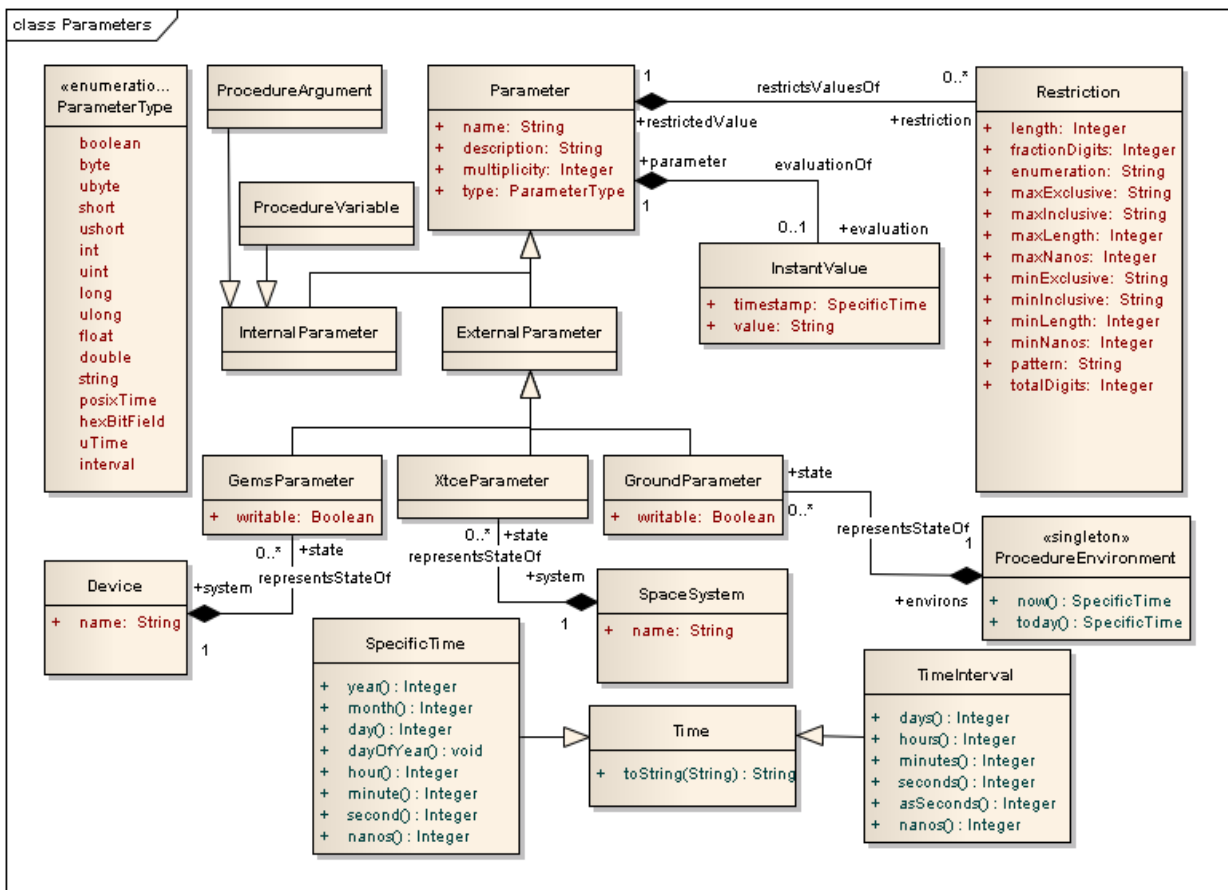
Figure 11 - GEMS and XTCE Parameters
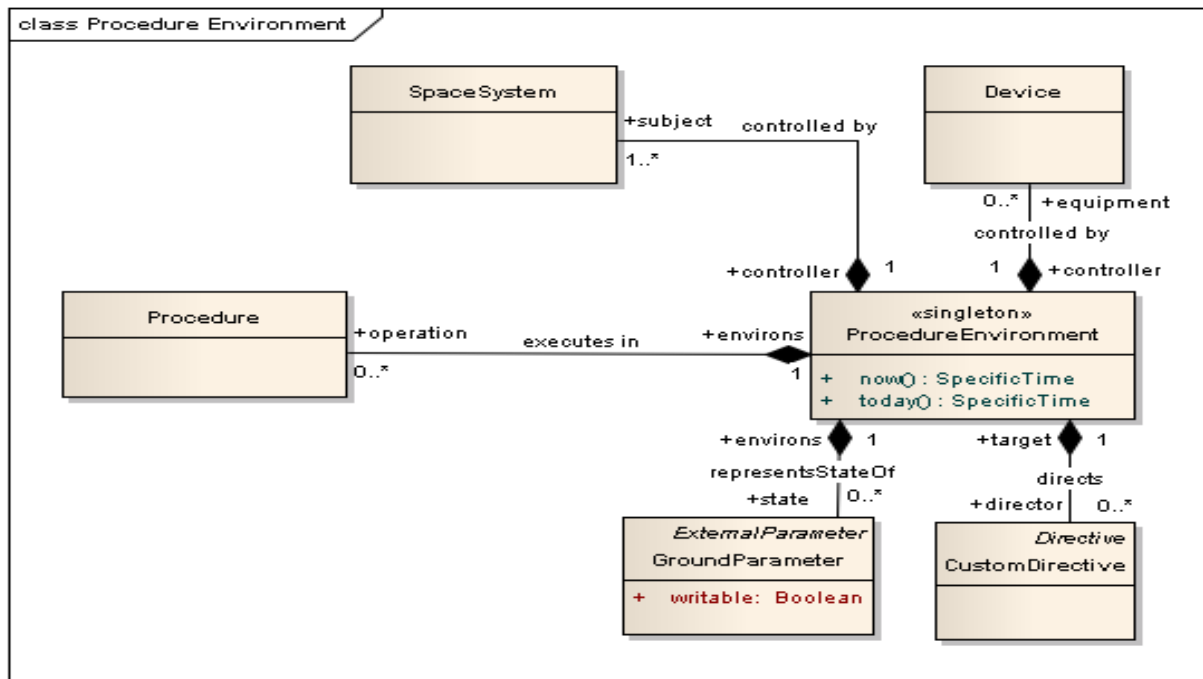
Figure 12 - Procedure Environment

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.

Generalizations

- "Time" on page 35

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints.

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.

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.

Generalizations

Attributes

- name:String - unique name for the device.

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.

Constraints

No additional constraints.

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.

Generalizations

- "Parameter" on page 30

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints.

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.

Generalizations

- "ExternalParameter" on page 28

Attributes

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

Associations

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

Constraints

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

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.

Generalizations

- "ExternalParameter" on page 28

Attributes

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

Associations

No additional associations.

Constraints

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

Semantics

See description above.

6.4.6 InstantValue

A value with a timestamp for a Parameter.

Generalizations

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.

Associations

- parameter:Parameter - the valued parameter.

Constraints

No additional constraints.

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.

Generalizations

- "Parameter" on page 30

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints.

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.

Generalizations

None

Attributes

No additional attributes.

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.

Constraints

No additional constraints.

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

Generalizations

None

Attributes

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

Associations

No additional associations.

Constraints

No additional constraints.

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.

Generalizations

- "Time" on page 35

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints.

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.

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

Generalizations

None

Attributes

No additional attributes.

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.

Constraints

No additional constraints.

Semantics

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

6.4.12 ProcedureVariable

Generalizations

- "InternalParameter" on page 30

Attributes

No additional attributes

Associations

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

Constraints

No additional constraints.

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.

Generalizations

None

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.

Associations

No additional associations.

Constraints

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

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.

Generalizations

None

Attributes

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

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.

Constraints

No additional constraints.

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

Generalizations

None

Attributes

None

Associations

No additional associations.

Constraints

No additional constraints.

Semantics

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

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 2.7 time module specification for time formatting.

6.4.16 XtceParameter

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

Generalizations

- "InternalParameter" on page 30

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints.

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.
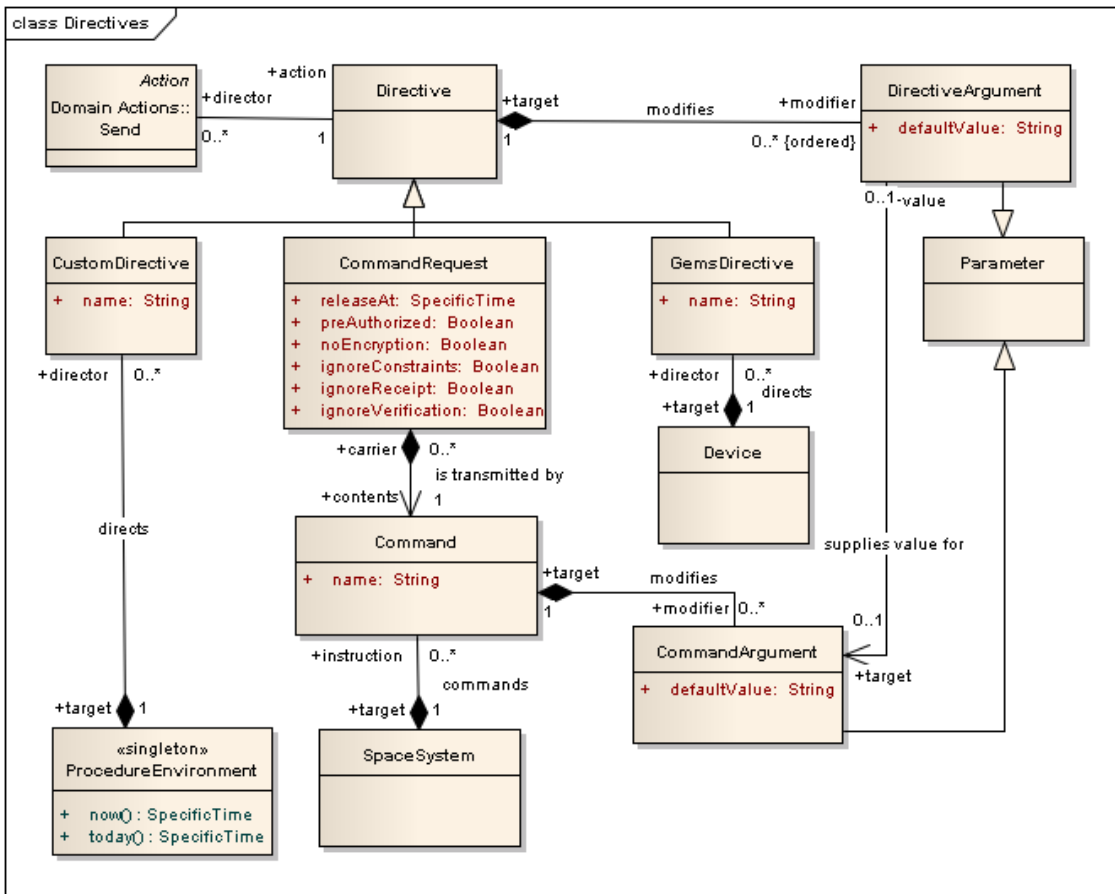
Figure 13 - Directives: CommandRequests and GemsDirectives

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.

Generalizations

None

Attributes

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

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.

Constraints

No additional constraints.

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.

Generalizations

- "Parameter" on page 30

Attributes

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

Associations

- modifier: 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.

Constraints

No additional constraints.

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.

Generalizations

- "Directive" on page 39

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.

Associations

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

Constraints

No additional constraints.

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 behavior is not directly transferable.

Generalizations

- "Directive" on page 39

Attributes

- name:String - the name of the directive.

Associations

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

Constraints

No additional constraints.

Semantics

The complete behavior 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.

Generalizations

None

Attributes

No additional attributes.

Associations

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

Constraints

No additional constraints.

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.

Generalizations

- "Parameter" on page 30

Attributes

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

Associations

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

Constraints

No additional constraints

Semantics

DirectiveArguments are defined during procedure definition and are passed to the ground system during Directive execution by the Send action. See page Error! Bookmark not defined..

6.5.7   GemsDirective

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

Generalizations

- "Directive" on page 39

Attributes

- name:String - the directive name

Associations

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

Constraints

No additional constraints

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.
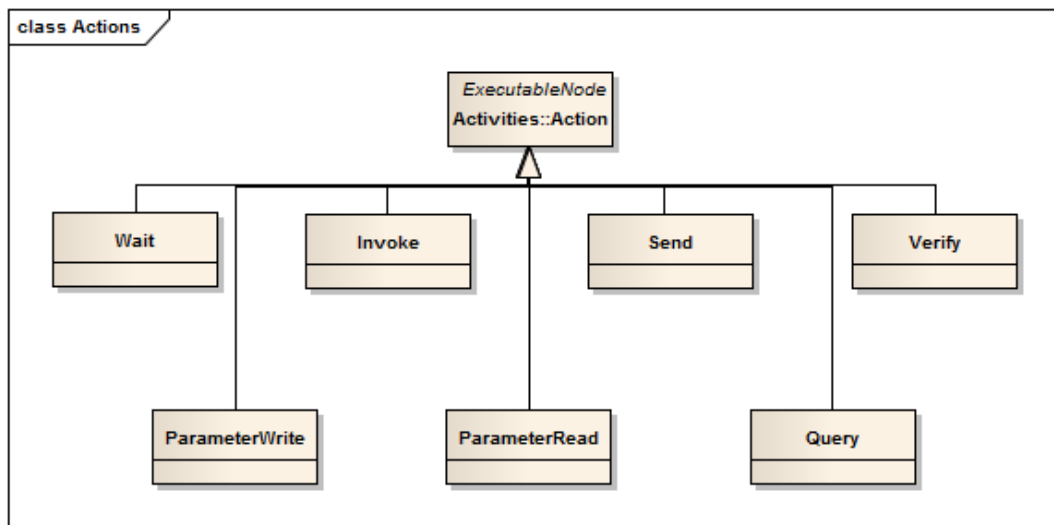
Figure 14 - SOLM Action Nodes for Activity Diagrams
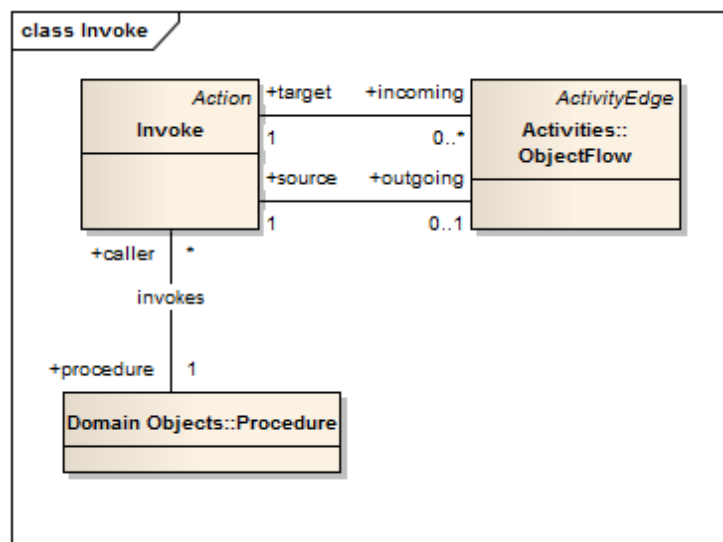
Figure 15 - Invoke Subprocedure

6.6.1   Invoke

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

Generalizations

- "Action" on page 15

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints

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.

Generalizations

- "Action" on page 15

Attributes

No additional attributes.

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.

Constraints

No additional constraints

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.

Generalizations

- "Action" on page 15

Attributes

No additional attributes.

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.

Constraints

No additional constraints

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.

Generalizations

- "Action" on page 15

Attributes

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

Associations

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

Constraints

No additional constraints

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.

Generalizations

- "Action" on page 15

Attributes

No additional attributes.

Associations

- outgoing: 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

Constraints

No additional constraints

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.

Generalizations

- "Action" on page 15

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints.

Semantics

See VerifyExpression and VerifyRange

6.6.7   VerifyExpression

VerifyExpression evaluates a Boolean expression.

Generalizations

- "Verify" on page 44

Attributes

- expression:String - string containing a python Boolean expression.

Associations

No additional associations.

Constraints

The expression must be a valid python Boolean expression (version 2.7). The syntax for python is maintained in an open source project at http://docs.python.org/reference.

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.

Generalizations

- "Verify" on page 44

Attributes

- expected:String - the expected value of the Parameter.

- tolerance:String - a plus/minus tolerance value to test for equality within a range.

Associations

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

Constraints

No additional constraints.

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.

Generalizations

- "Action" on page 15

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints.

Semantics

See WaitOnExpression and WaitOnTime for specific semantics.

6.6.10 WaitOnExpression

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

Generalizations

- "Wait" on page 46

Attributes

- pollPeriod:RelativeTime - the time period to wait before re-evaluating the expression.

- timeout:Time - the time period or AbsoluteTime to wait before failing due to timeout

Associations

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

Constraints

No additional constraints.

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.

Generalizations

- "Wait" on page 46

Attributes

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

Constraints

No additional constraints.

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.
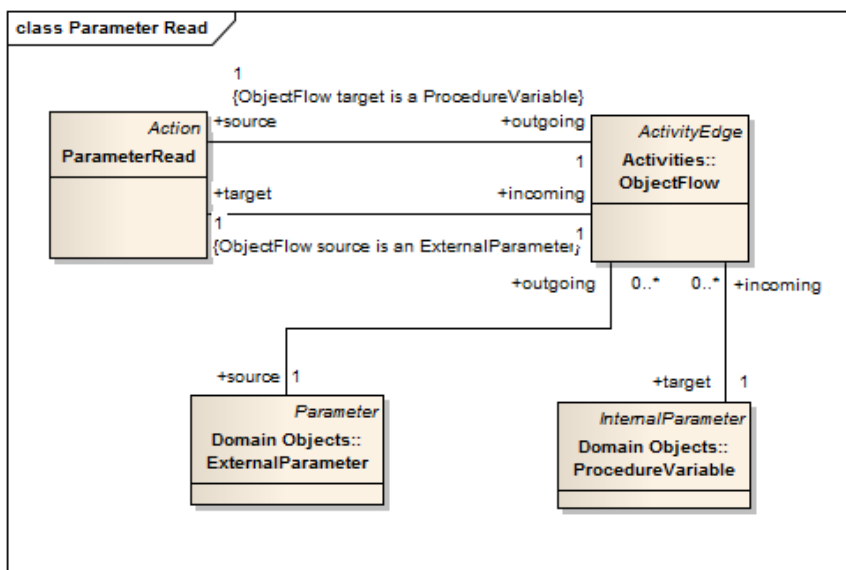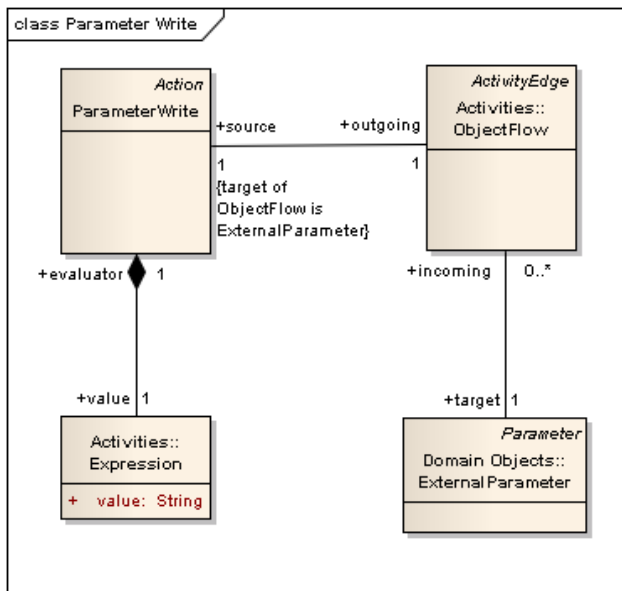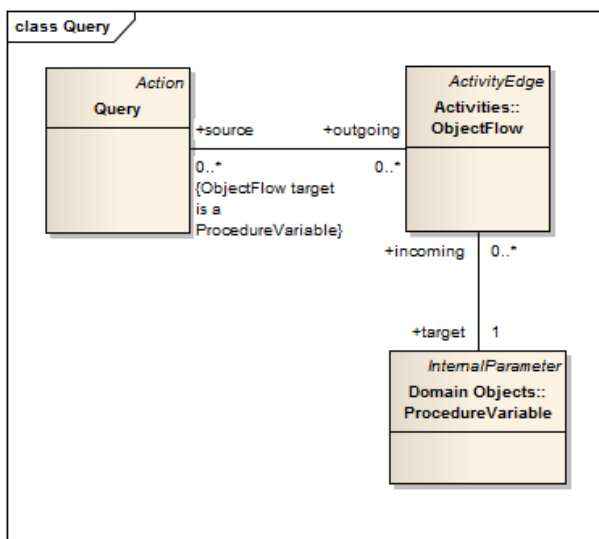

Figure 16 - Parameter Read

Figure 17 - Parameter Write

Figure 18 - Query Operator
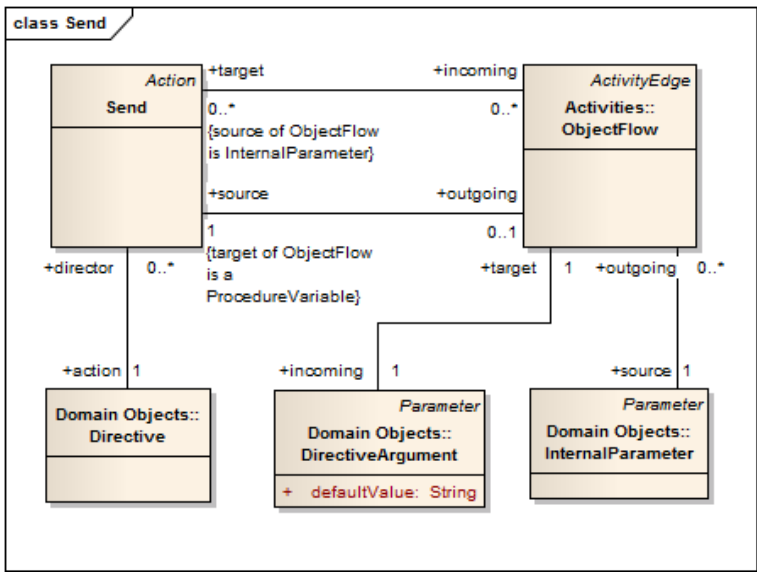
Figure 19 - Send Directive
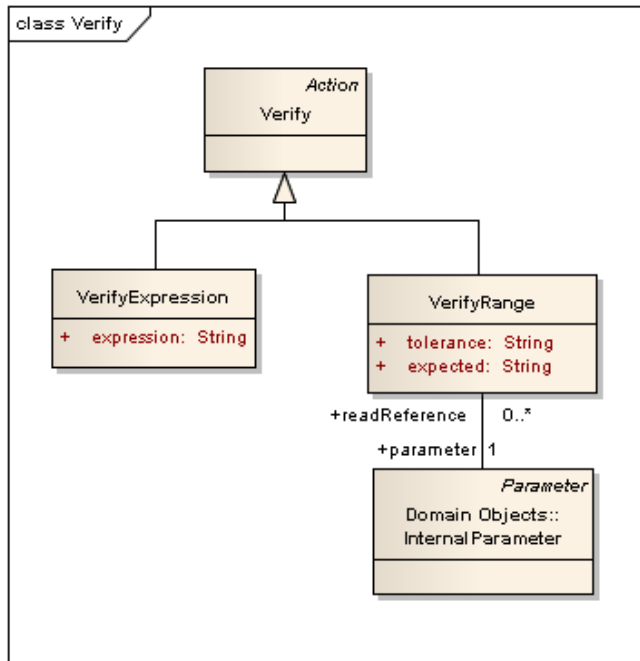
Figure 20 - Verify State
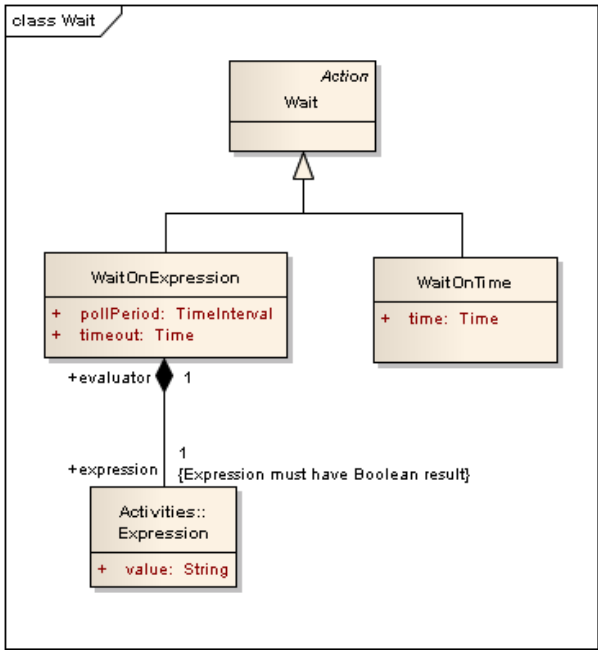
Figure 21 - Wait

7   Comet Control Language (CCL) Mapping

7.1   General

The native scripting language for Harris Corporation's OS/COMET® product is known as Comet Control Language, or CCL. This section provides information on mapping a procedure defined in SOLM into a CCL procedure. In the mapping table, a value inside <> indicates a named entity that is used by name in the procedure. A CCL representation of the procedure shown in Figure 4 on page 6 is provided below:

```
PARAMDEF NPARAM=SpeedIncrement
ATTACH SAT="SAT_1"
IF (MomentumWheelState .EQ. "Off") THEN
CEXL MomentumWheelOn
ENDIF
CEXL SetWheelSpeed, WheelSpeed=MomentumWheelSpeed+'SpeedIncrement'
```

Table 1 - CCL Constructs for SOLM Mapping

CCL	SOLM	Comments
! or ;	Comment	An activity diagram note in the procedure sequence
DOWHILE..ENDDO	DecisionNode, MergeNode	CCL also supports an iterative DO..ENDDO and REPEAT..UNTIL constructs
INQUIRE	Query	
PAUSE	Query with no Parameters	
ONERROR	HandledExceptionRegion	An ONERROR statement sets up an exception handler for errors that occur during procedure execution, including an error in the Verify statement.
VERIFY	Verify	A statement that accepts a Boolean expression and transfers control to an exception handler, if one is defined, or halts the procedure if no exception handler is provided.
ATTACH <SPACESYSTEM_NAME>	SpaceSystem	Associates command and telemetry references with the SpaceSystem specified by unique name.

PARAMDEF	Procedure, ProcedureArgument	Defines the named parameters for a procedure
PERFORM <PROCEDURE_NAME>	Invoke	Invokes the specified procedure or sub-procedure.
IF..[ELSE]..ENDIF	DecisionNode	Defines a conditional execution path.
CEXL <COMMAND_NAME>	CommandRequest Command	Requests command transmission for the command specified by unique name with the specified keyword values for the SpaceSystem referred to in the most recent ATTACH statement.
<Parameter>	Parameter ParameterRead	A parameter value is referenced by using its unique name in an expression
SET <Parameter> = expression	Parameter ParameterWrite	A parameter value may be overridden by a SET statement. This override replaces a local copy and does not change the value in the associated SpaceSystem or GEMS device
GEMS <device>, GET, PARAMS=<param1&param2…>	GemsParameter ParameterRead	Get the current values for the specified Parameters in the specified GEMS device.
GEMS <device>, SET, PARAMS=<param1&param2…>, VALUES=<value1&value2…>	GemsParameter ParameterWrite	Set the current values for the specified Parameters in the specified GEMS device.
GEMS <device>, <directive>,PARAMS=<param1&param2…>, VALUES=<value1&value2…>	GemsDirective	Issue the specified directive to the specified GEMS device with the specified Parameter values.

SOLM	CCL	Comments
Procedure, ProcedureArgument	PARAMDEF NPARAM=<parameters defined in Signature>	If there are no input parameters, the PARAMDEF is not required in CCL.
SpaceSystem	ATTACH SAT="<Link.SpaceSystem>"	A Link must be defined in the procedure class model in order to access XTCE Parameters and Commands

Device		No statement is required in CCL until a parameter is read, a parameter is written, or a directive is issued to the GEMS Device.
DecisionNode	IF <guard1 expression> … ELSEIF<guard2 expression> … ELSE <default guard> … ENDIF	The guard paths should be exclusive, since there is no order implied in the activity diagram other than the default guard.
MergeNode	ENDIF or ENDDO	There is no concept of threads within CCL, so the merge node would typically only be seen at the end of a conditional path or loop.
ParameterRead	<Parameter.name> Or GEMS <Device.name>,GET, PARAMS=<Parameter.name>	In CCL, the parameter name may be used in an expression to retrieve the current value for an XTCE Parameter. Reading a GemsParameter requires a specific directive.
ParameterWrite	SET <Parameter.name> Or GEMS <Device.name>,SET, PARAMS=<Parameter.name>, VALUES=<Value>	
Send	CEXL <Command.name> or GEMS <GemsDirective.name>	The Send action is translated based on the Directive type.
ParameterWrite	SET <Variable.name> = <Expression>	The syntax of arithmetic expressions in CCL is similar to Python, but Boolean and logical operators are FORTRAN format in CCL.
ActivityFinalNode	EXIT <status>	
HandledExceptionRegion	ONERROR GOSUB <Handler Label> <HandlerLabel>:	

8   SpacePython Mapping

8.1   General

The 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 chapter provides information on mapping SOLM to a SpacePython script. SpacePython is based on the Python 2.7 syntax with the addition of the SOLM extensions defined in this chapter to support spacecraft operations. A SpacePython representation of the procedure shown in Figure 4 on page 6 is provided below:

```
"""SetMomentumWheelSpeed
    This SpacePython module provides an example of a spacecraft
 operations procedure that checks a subsystem state, optionally
 issues a corrective command, and then sends a command to set
 momentum wheel speed based on a change to the current speed.
"""
from space import Link
def invoke(**kwargs):
    """ Change the current momentum wheel speed by the positive
        or negative rpm specified by the keyword parameter,
        SpeedIncrement
    """
sat1 = Link('SAT_1')
    MomentumWheelState = sat1.lookupParameter( 'MomentumWheelState')
    MomentumWheelSpeed = sat1.lookupParameter('MomentumWheelSpeed')
    if MomentumWheelState.value() == 'Off':
    sat1.send('MomentumWheelOn')
SetWheelSpeed = sat1.lookupCommand('SetWheelSpeed')
SetWheelSpeed.WheelSpeed = MomentumWheelSpeed.value()+kwargs['SpeedIncrement']
sat1.send(SetWheelSpeed)
```

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. This is in the Python 2.7 base language.

If..[elif]..[else]	DecisionNode	Defines conditional execution paths. Indentation is significant for defining the span of each conditional block. This is in the Python 2.7 base language.
Invoke	InitialNode	Defines the primary entry point for a procedure accepting keyword arguments for the parameter values. 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 def operatorQuery(prompt, parameterList): 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. This is a SpacePython library function
Link(<SPACESYSTEM>)	SpaceSystem	A Link object represents a command and telemetry connection through the ground system to a specific SpaceSystem defined by an XTCE document. The Link is created in a procedure using the unique name of the SpaceSystem. If the underlying system needs additional information or settings to establish the Link, these may be supplied in a system-specific way using the SpaceSystem name as a key. This is a SpacePython class definition.
Link.send(<COMMAND>)	CommandRequest Command	The send method of a Link object issues a command transmission request to the ground system for the SpaceSystem associated with the Link. A fully-specified command (no required Arguments) can be specified by its unique name, otherwise a Command instance must be obtained from the Link.lookupCommand() method and completed by specifying required arguments to the Command instance. This is a SpacePython class method.
Link.lookupParameter (<PARAMETER>)	XtceParameter ParameterType Restriction	Obtain an instance of a Parameter in the SpaceSystem associated with the Link (or Downlink). This is a SpacePython class method.

Link.lookupCommand (<COMMAND>)	Command	Obtain a reference to a Command in the SpaceSystem associated with the Link. This is a SpacePython class method.
Downlink(<SPACESYSTEM>)	SpaceSystem	A Downlink object represents a telemetry-only connection to a SpaceSystem. Telemetry Parameters may be looked up and read but cannot be written. A Downlink object also does not support the send() method for CommandRequests. This is a SpacePython class definition.
GemsDevice(<DEVICE>)	Device	A GemsDevice object represents a connection to a specific GEMS 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.
GemsDevice.get()	GemsParameter ParameterRead	Obtain the current values of a list of Parameters, specified by unique names, from a GEMS device. This is a SpacePython class method.
GemsDevice.set()	GemsParameter ParameterWrite	Set the values of a list of Parameters in a GEMS device, specified as a list of name=value pairs. This is a SpacePython class method.
GemsDevice.lookupDirective()	GemsDirective Restriction	Obtain an instance of a Directive, specified by unique name, defined for the GEMS device. This is a SpacePython class method.
GemsDevice.send()	GemsDirective	Issues a Directive to the GEMS device. The Directive instance must be obtained by the GemsDevice.lookupDirective() method and completed by supplying any required Parameter values. This is a SpacePython class method.
<Parameter>	Parameter	Represents an XTCE or GEMS Parameter. This is a SpacePython abstract class.
<Parameter>__<Attribute>	Restriction	Provides the attributes of a parameter including limits and valid range.
Parameter.value()	XtceParameter ParameterRead	Returns the engineering unit value of the Parameter for use in an expression. 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. 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.
GemsParameter	GemsParameter	Represents a GEMS Parameter definition. Must be obtained by a factory method "lookupParameter" on the GemsDevice. This is a SpacePython class.
XtceParameter	XtceParameter	Represents an XTCE Parameter definition. Must be obtained by a factory method "lookupParameter" on Link or Downlink.

SOLM	SpacePython	Comments
InitialNode	`from space import Link` `def invoke(**kwargs):`	Python accepts param=value style keyword arguments.
SpaceSystem	link = Link("<Link.SpaceSystem>")	A Link must be defined in the procedure class model in order to access XTCE Parameters and Commands
Device	dev = GemsDevice("<Device.name>")	A Device must be defined in the procedure class model in order to access GEMS Parameters and Directives

DecisionNode	if <guard1 expression>:
	…
	elif<guard2 expression>:
	...
	else <default guard>:
	...
		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.get('<Parameter.name>')	
ParameterWrite	dev.set(<Parameter.name>=value)	
Send	dev.send(directive) or link.send(command)	The Send action is translated based on the Directive type.
ParameterWrite	<Variable.name> = <Expression>	

ActivityFinalNode	return	

HandledExceptionRegion	try:
	    …
	catch:
	    …	