Constraints

The Structure Language may sometimes be insufficient to express advanced constraints on the language structure. The Constraints aspect gives you a way to define such additional constraints.

Can be child/parent/ancestor/root

These are the first knobs to turn when defining constraints for a concept. They determine whether instances of this concept can be hooked as children (parents, ancestors) nodes of other nodes or root nodes in models.You specify them as boolean closures to invoke each time when evaluating allowed possition for a node in the AST.

can be child

Return false if an instance of the concept is not allowed to be a child of specific nodes.

parameter	description
operationContext	IOperationContext
scope	current context (IScope)
parentNode	the parent node we are checking
childConcept	concept of the child node (can be a subconcept of this concept)
link	LinkDeclaration of the child node (child role can be taken from there)

can be parent

Return false if an instance of concept is not allowed to be a parent of specific concept node (in a given role).

parameter	description
operationContext	IOperationContext
scope	context (IScope)
node	the parent node we are checking (instance of this concept)
childConcept	the concept of the child node we are checking
link	LinkDeclaration of the child node

can be ancestor

Return false if an instance of the concept is not allowed to be an ancestor of specific nodes.

parameter	description
operationContext	IOperationContext
scope	context (IScope)
node	the ancestor node we are checking (instance of this concept)
childConcept	the concept of the descendant node

can be root

This constraint is available only for rootable concepts (instance can be root is true in the concept structure description). Return false if instance of concept cannot be a root in the given model.

parameter	description
operationContext	IOperationContext

scope	context (IScope)
model	model of the root

Property constraints

Technically speaking, "pure" concept properties are not properties in its original meaning, but only public fields. Property constraints allow you to make them real properties. Using these constraints, the behavior of concept's properties can be customized. Each propertz constraint is applied to a single specified property.

property - the property to which this constraint is applied.

get - this method is executed to get property value every time property is accessed.

parameter	description
node	node to get property from
scope	context (IScope)

set - this method is executed to set property value on every write. The property value is guaranteed to be valid.

parameter	description
node	node to set property
propertyValue	new property value
scope	context (IScope)

is valid - this method should determine whether the value of the property is valid. This method is executed every time before changing the value, and if it returns false, the set() method is not executed.

parameter	description
node	node to check property
propertyValue	value to be checked
scope	context (IScope)

Referent constraints

Constraints of this type help to add behavior to concept's links and make them look more properties-like.

referent set handler - if specified, this method is executed on every set of this link.

parameter	description
referenceNode	node that contains link.
oldReferentNode	old value of the reference.
newReferentNode	new value of the reference.
scope	context: IScope interface to object that shows you models, languages and devkits you can see from the code.

search scope - defines the set of nodes to which this link can point. The method can return either a sequence<node<>> or ISe archScope.

parameter	description
model	the model that contains the node with the link. This is included for convenience, since both referenceNode and enclosingNode keep the model too.

scope	IScope interface that shows models, languages and devkits you can see from the code.
referenceNode	the node that contains the actual link. It can be null when a new node is being created for a concept with smart reference. In this situation smart reference is used to determine what type of node to create in the context of enclosingNode, so the search scope method is called with a null referenceNode.
enclosingNode	parent of the node that contains the actual link, null for root nodes. Both referenceNode and enclosingNode c annot be null at the same time.
linkTarget	the concept that this link can refer to. Usually it is a concept of the reference, so it is known statically. If we specialize reference in subconcept and do not define search scope for specialized reference, then linkTarget parameter can be used to determine what reference specialization is required.

If search scope is not set for the reference then default scope from the referenced concept is used. If the default search scope is also not set then "global" scope is used: all instances of referent concept from all imported models.

validator (since 1.5) - Each reference is checked against it's search scope and if, after changes in the model, a reference ends up pointing out of the search scope, MPS marks such a reference with an error message. Sometimes it is not efficient to build the whole search scope just to check if the reference is in scope. The search scope can be big or it may be much easier to check if the given node is in scope than to calculate what nodes are in scope. You can create quick reference check procedure here to speed up reference validation in such situations.

parameter	description
model scope referenceNode enclosingNode linkTarget	context of reference usage, the same meaning as in the search scope method. The main difference: referenceNode cannot be null here because validator is not used during node creation.
checkedNode	the node to be validated (referenceNode has a reference to checkedNode of type linkTarget)

It is possible to create validation routine only if you have created a list of nodes in the corresponding search scope. If ISearchScope is returned from search scope method, then isInScope(SNode) method of ISearchScope interface will be used for validation, you should override this method with your validation routine.
It is not possible to define validation routine without defining search scope.

presentation - here you specify how the reference will look like in the editor and in the completion list. Sometimes it is convenient to show reference differently depending on context. For example, in Java all references to an instance field f should be shown as this.f, if the field is being shadowed by the local variable declaration with the same name. By default, if no presentation is set, the name of the reference node will be used as its presentation (provided it is an INamedConcept).

parameter	description
model scope referenceNode enclosingNode linkTarget	the context of reference usage, the same meaning as in the search scope function.
parameterNode	the node to be presented (referenceNode has a reference to parameterNode of type linkTarget)
visible	true - presentation of existing node, false - for new node (to be created after selection in completion menu)
smartReference	true - node is presented in the smart reference
inEditor	true - presentation for editor, false - for completion menu

ISearchScope

Low level interface that can be implemented to support search scope. We recommend to subclass `AbstractSearchScope` (implements `ISearchScope`) abstract class instead of direct implementation of `ISearchScope` interface. The only abstract method in `AbstractSearchScope` class is
`@NotNull List<SNode> getNodes(Condition<SNode> condition)` - return list of nodes in the current search scope satisfying `condition`, similar to search scope but with the additional condition.

Other useful methods to override:
`boolean isInScope(SNode node)` - the same function as in validator method.
`IReferenceInfoResolver getReferenceInfoResolver(SNode, AbstractConceptDeclaration);`

resolve info - TODO

Default scope

Suppose we have a link pointing to an instance of concept C and we have no #search scope defined for this link in referent constraints. When you edit this link, all instances of concept C from all imported models are visible by default. If you want to restrict set of visible instances for all links to concept C you can set default scope for the concept. As in referent constraint you can set search scope, validator and presentation methods. All the parameters are the same.

Previous Next