## Contents

Preface ........................................................................................................ ix  
1 Introduction ................................................................................................. 1  
  1.1 Scope and Purpose of this Document ......................................................... 1  
  1.2 Prerequisite Knowledge ........................................................................... 1  
  1.3 Motivation for Using MapleMBSE Virtual Features ................................. 1  
  1.4 Importing the MapleMBSE Ecore .............................................................. 3  
  1.5 General Syntax for the MapleMBSE Virtual Features ............................. 3  
2 Stereotypes .................................................................................................. 5  
  2.1 metaclassName ........................................................................................ 5  
    Description ................................................................................................ 5  
    Syntax ....................................................................................................... 5  
    Using the metaclassName Virtual Feature .................................................. 5  
    Example .................................................................................................... 6  
  2.2 featureName .......................................................................................... 6  
    Description ................................................................................................ 6  
    Syntax ....................................................................................................... 7  
    Using the featureName Virtual Feature ..................................................... 7  
    Example .................................................................................................... 8  
  2.3 stereotypeNames .................................................................................... 9  
    Description ................................................................................................ 9  
    Syntax ....................................................................................................... 9  
    Using the stereotypeNames Virtual Feature .............................................. 9  
    Example .................................................................................................... 10  
3 Associations ............................................................................................... 11  
  3.1 associatedProperty ................................................................................ 11  
    Description ................................................................................................ 11  
    Syntax ....................................................................................................... 11  
    Using the associatedProperty Virtual Feature ........................................... 12  
    Example .................................................................................................... 13  
  3.2 directedAssociatedProperty .................................................................... 14  
    Description ................................................................................................ 14  
    Syntax ....................................................................................................... 14  
    Using the directAssociatedProperty Virtual Feature ............................... 14  
    Example .................................................................................................... 15  
  3.3 otherAssociatedEnd .............................................................................. 16  
    Description ................................................................................................ 16  
    Syntax ....................................................................................................... 16  
    Using the otherAssociatedEnd Virtual Feature ....................................... 16  
    Example .................................................................................................... 17  
  3.4 nestedDirectedComposition ................................................................... 19  
    Description ................................................................................................ 19
## List of Figures

<table>
<thead>
<tr>
<th>Figure Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 2.1: metaclassName Example</td>
<td>6</td>
</tr>
<tr>
<td>Figure 2.2: A The appliedStereotypeInstance Structure</td>
<td>7</td>
</tr>
<tr>
<td>Figure 2.3: featureName Example</td>
<td>8</td>
</tr>
<tr>
<td>Figure 2.4: stereotypeNames Example</td>
<td>10</td>
</tr>
<tr>
<td>Figure 3.1: associatedProperty Example</td>
<td>13</td>
</tr>
<tr>
<td>Figure 3.2: directAssociatedProperty Example</td>
<td>15</td>
</tr>
<tr>
<td>Figure 3.3: otherAssociatedEnd Example</td>
<td>18</td>
</tr>
<tr>
<td>Figure 5.1: connectedPropertyOrPort Example</td>
<td>28</td>
</tr>
<tr>
<td>Figure 5.2: otherConnectorEnd Example</td>
<td>30</td>
</tr>
<tr>
<td>Figure 6.1: clientDependencies Example</td>
<td>32</td>
</tr>
<tr>
<td>Figure 6.2: supplierDependencies Example</td>
<td>34</td>
</tr>
<tr>
<td>Figure 8.1: multiplicityProperty Example</td>
<td>40</td>
</tr>
<tr>
<td>Figure 9.1: ActivityControlFlow Example</td>
<td>42</td>
</tr>
<tr>
<td>Figure 9.2: ActivityObjectFlow Example</td>
<td>44</td>
</tr>
<tr>
<td>Figure 10.1: VertexTransition Example</td>
<td>46</td>
</tr>
</tbody>
</table>
Preface

MapleMBSE Overview

MapleMBSE™ gives an intuitive, spreadsheet based user interface for entering detailed system design definitions, which include structures, behaviors, requirements, and parametric constraints.

Related Products

MapleMBSE 2021 requires the following products.

- Oracle® Java® SE Runtime Environment 8.

Note: MapleMBSE looks for a Java Runtime Environment in the following order:

1) If you use the -vm option specified in OSGiBridge.init (not specified by default), MapleMBSE will use it.

2) If your environment has a system JRE (meaning either: JREs specified by the environment variables JRE_HOME and JAVA_HOME in this order, or a JRE specified by the Windows Registry (created by JRE installer) ), MapleMBSE will use it.

3) The JRE installed in the MapleMBSE installation directory.

If you are using IBM® Rational® Rhapsody® with MapleMBSE, the following versions are supported:

- Rational Rhapsody Version 8.1.5, 8.3 and 8.4
- Teamwork Cloud™ server 18.5 SP3 or 19.0 SP4

If you are using Eclipse Capella™ with MapleMBSE, the following version is supported:

- 1.4.0

If you are using Eclipse™, the following version is supported:

- 2020-3

Note that the architecture of the supported non-server products (that is, 32-bit or 64-bit) must match the architecture of your MapleMBSE architecture.
Related Resources

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MapleMBSE Installation Guide</td>
<td>System requirements and installation instructions for MapleMBSE. The <strong>MapleMBSE Installation Guide</strong> is available in the <code>Install.html</code> file located either on your MapleMBSE installation DVD or the folder where you installed MapleMBSE.</td>
</tr>
<tr>
<td>MapleMBSE Applications</td>
<td>Applications in this directory provide a hands on demonstration of how to edit and construct models using MapleMBSE. They, along with an accompanying guide, are located in the Application subdirectory of your MapleMBSE installation.</td>
</tr>
<tr>
<td>MapleMBSE Configuration Guide</td>
<td>This guide provides detailed instructions on working with configuration files and the configuration file language.</td>
</tr>
<tr>
<td>MapleMBSE User Guide</td>
<td>Instructions for using MapleMBSE software. The <strong>MapleMBSE User Guide</strong> is available in the folder where you installed MapleMBSE.</td>
</tr>
<tr>
<td>Frequently Asked Questions</td>
<td>You can find MapleMBSE FAQs here: <code>https://faq.maplesoft.com</code></td>
</tr>
<tr>
<td>Release Notes</td>
<td>The release notes contain information about new features, known issues and release history from previous versions. You can find the release notes in your MapleMBSE installation directory.</td>
</tr>
</tbody>
</table>

For additional resources, visit [http://www.maplesoft.com/site_resources](http://www.maplesoft.com/site_resources).

**Getting Help**

To request customer support or technical support, visit [http://www.maplesoft.com/support](http://www.maplesoft.com/support).

**Customer Feedback**

Maplesoft welcomes your feedback. For comments related to the MapleMBSE product documentation, contact doc@maplesoft.com.
1 Introduction

1.1 Scope and Purpose of this Document

The purpose of the MapleMBSE Virtual Features Guide is to describe MapleMBSE virtual features and explain how to use them.

The intended audience for this document are users who are familiar with UML, SysML and Model-based Systems Engineering concepts and who intend to create their own MapleMBSE configuration files.

1.2 Prerequisite Knowledge

To fully understand the information presented in this document the reader should be familiar with the following concepts:

• The Eclipse Modeling Framework ecore serialization. In particular, knowing how to use any tool of your choice to track all the eReferences independently of the eSuperTypes.

• Thus, some basic concepts of Meta Object Facility like eClassifiers and eStructuralFeatures. A correct mse configuration file has within each qualifier a concrete UML eClassifiers and each dimension should be accessed using a non-derived StructuralFeature defined in the UML.ecore or a virtual one inside this guide.

• MapleMBSE Configuration Language elements (especially dimension and qualifiers, and the syntax for importing the MapleMBSE ecore). For more information on the MapleMBSE Configuration language, see the MapleMBSE Configuration Guide.

1.3 Motivation for Using MapleMBSE Virtual Features

SysML provides a high level of abstraction to cover as many modeling scenarios as possible with the diagrams offered. It is a powerful and complex language that is extremely difficult to master because of its complexity (there are hundreds of pages of technical specifications for SysML).

Many different concrete and abstract Classifiers, with very specific semantics, are part of the SysML technical specifications. These Classifiers should not be used interchangeably. Even "linking" elements changes depending on the "linked" elements. For example, SysML Associations are to Classes as Connectors are to Ports, or, what ControlFlows can be for ActivityNodes. However, these elements are not interchangeable.

An end user, defined as a user who will be updating model information using the MapleMBSE spreadsheet interface but likely will not be involved in creating or editing configuration files, who interested in taking advantage of the modeling capabilities of SysML, should not need to know its complexities. MapleMBSE helps to hide this complexity.
from the end user, through virtual features. They are called virtual features because, although they extend the capabilities of native SysML, they themselves are not part of SysML.

With the right choice of labels within an Excel template and a well designed configuration (.mse) file that implements MapleMBSE virtual features, an end user can enter a couple of inputs in a spreadsheet and create Blocks and the Associations linking them, or Ports and Connectors, or other combinations of elements.

For example, consider the following code snippet from a MapleMBSE configuration file in the figure below. This figure illustrates the scenario where a configuration file is designed without the use of virtual features to represent SysML Associations between Blocks.

Notice in the generated Excel worksheet, the number of inputs required of the end user to represent the Association between Customer and Product. This requires knowledge of SysML on the part of the end user.

Now consider an example that represents the same Association between Customer and Product, as shown in the figure below. This time, the configuration file is designed using the MapleMBSE virtual features, specifically, the associatedProperty virtual feature. Notice, the only inputs required of the end user are the two SysML Blocks, Customer and Product. The cross-references required for the Association are completed automatically.
1.4 Importing the MapleMBSE Ecore

Loading MapleMBSE virtual features is analogous to the way you would load UML Structural Features using UML Ecore. The corresponding MapleMBSE Configuration language uses `import-ecore`.

The general syntax is

`import-ecore "URI"`

For example, to specify the NoMagic ecore:

"http://www.nomagic.com/magicdraw/UML/2.5"

To specify the MapleMBSE ecore:

"http://maplembse.maplesoft.com/common/1.0"

You must create an alias for the ecore using the syntax:

`import-ecore "URI" as Alias`

For example, to specify an alias for the MapleMBSE ecore:

`import-ecore "http://maplembse.maplesoft.com/common/1.0" as mse`

This allows you to use the short form, `mse`, instead of the whole syntax.

1.5 General Syntax for the MapleMBSE Virtual Features

The general syntax for the virtual features is

`[/.]?alias::virtualfeature`
The first character can be a dot, a forward slash, or a blank. There is no strict rule of thumb for this. For specific syntax, see the Syntax subsection for each virtual feature.

**alias** - This is the alias for the ecore import

**virtualfeature** - This is the virtual feature name you want to use, for example, `associatedProperty`. 
2 Stereotypes

SysML can be explained as a subset of elements defined in the UML specifications plus some additional features not included in UML. One of these features is a **Stereotype**. Stereotypes are applied to those elements adding extra meaning or modeling semantics. MapleMBSE offers several virtual features to apply Stereotypes and navigate their extended modeling capacities.

2.1 metaclassName

**Description**

Use the **metaclassName** virtual feature to apply Stereotypes while creating elements using MapleMBSE. To use this virtual feature you need to identify the qualified name of the Stereotype that you want to apply and whether the element is compatible with that stereotype.

**Syntax**

Any **Element** of the **Model** can have a list of **appliedStereotype** but only certain Stereotypes should be applied to certain **Element**. This is one of the few virtual features that is used as a filter inside the qualifier and it does not require a dot or slash notation prior to the alias. The **metaclassName** virtual feature must be followed by an equals symbol and the qualified name of the Stereotype between quotation marks.

alias::metaclassName="qualified::name"

It is important to note that this qualified name is basically a path and the name that identifies uniquely each Stereotype, and each substring is concatenated with a double colon notation.

**Using the metaclassName Virtual Feature**

The following steps illustrate what you need to do to use the metaclassName virtual feature:

1. The MapleMBSE.ecore is imported and its alias is mse.

2. Two data-sources are used for this example with **metaclassName** to filter **Blocks** and **Requirements**. **Note**: both of those SysML concept are UML Classes but with different Stereotypes.

3. Defining **synctable-schemas**, one for **Blocks** and another for **Requirements**. **Note**: To avoid problems with MapleMBSE it is a good practice to use the same qualifier and Stereotype filter in the data-source and the first dimension of the schema.

4. Complete the rest of the configuration as usual: **worksheet-templates**, **synctable** and **workbook**.
Example

The following example showcases how to use `metaclassName` to create `Classes` applying 2 different `Stereotypes`.

```java
import ecore "http://www.omg.com/magicdraw/UML/2.5"
import ecore "http://maplembse.maplesoft.com/common/1.0" as mse

data-source Root[Model]
data-source blocks = Root/packageElement[Class | mse:metaclassName="SysML::Blocks::Block"]
data-source requirements = Root/packageElement[Class | mse:metaclassName="SysML::Requirements::Requirement"]

// symtable-schema BlockSchema {
  // record din [Class | mse:metaclassName="SysML::Blocks::Block"] {
  //   key column /name as bname
  //   ...
  // }
  //
  // symtable-schema RequirementSchema {
  // record din [Class | mse:metaclassName="SysML::Requirements::Requirement"] {
  //   key column /name as rname
  //   ...
  // }
  //
  // worksheet-template BlockTemplate (bs: BlockSchema) {
  //   vertical table tab1 at (1, 1) = bs {
  //     ...
  //   }
  // }
  //
  // worksheet-template RequirementTemplate (rsc: RequirementSchema) {
  //   vertical table tab2 at (1, 1) = rsc {
  //     ...
  //   }
  // }
  //
  // symtable blockTable = BlockSchema(blocks)
  // symtable requirementTable = RequirementSchema(requirements)
  //
  // workbook {
    // worksheet BlockTemplate(blockTable)
    // worksheet RequirementTemplate(requirementTable)
  // }
```

Figure 2.1: `metaclassName` Example

2.2 `featureName`

Description

As mentioned in the introduction of this section, once you applied a `Stereotype` to any `Element`, you are changing its semantics and extending it. Use `featureName` to access those extended properties stored in `Slots` using their qualified names.

The class diagram in Figure 2.2 (page 7) shows the different `EClasses` that need to be queried in order to access those `Slots`, remember that `Element` is an abstract `EClass` and it should not be used as the qualifier. Basically all elements in a `Model` implement `Element`, thus `EClasses` like `Class` have the structural feature `appliedStereotypeInstance` to query `InstanceSpecification`. 
Figure 2.2: A The appliedStereotypeInstance Structure

Syntax

Use `featureName` the same way `metaclassName` is used within a qualifier as a filter, meaning that no dot or slash notations are needed before the alias. It is expected, following the virtual feature, an equal symbol and a string between quotation marks; this string is the qualified name of the property to access.

```
alias::featureName="qualified::name"
```

This qualified name is similar to the one used to identify the `Stereotype` but it differs slightly at the end with extra information concatenated to identify a single extension. As mentioned before this virtual feature is usable while querying a `Slot` inside an `InstanceSpecification` inside an concrete `Element`, but you must also know that this `Element` must be filtered by `metaclassName` with the qualified name that identifies the `Stereotype`.

Using the `featureName` Virtual Feature

To access extra Properties added after applying a Stereotype:

1. Import the MapleMBSE ecore.
2. Inside a synctable-schema navigate to a `MultiplicityElement`, in this case, `/ownedAttribute[Property]` within a `Class`.
3. Within that dimension, define a regular column using `/mse::multiplicityProperty`.
4. Complete the rest of the configuration as usual: worksheet-templates, synctable and workbook.
Example

The following example illustrates how to access extra Properties added after applying a Stereotype.

1. Import MapleMBSE ecore, for this example use mse as the alias.
2. Create a data-source using the metaclassName virtual feature mentioned before to filter Requirements.
3. Define a synctable-schema for Requirements. Note: use the same qualifier and Stereotype for the first dimension as for the data-source.
4. To access the SysML::Requirements::Requirement::Text Property added to a Class after applying the Requirement Stereotype you must:
   1. Navigate appliedStereotypeInstance to get an InstanceSpecification.
   2. Then slot to recover all the Slots within the InstanceSpecification
   3. Use featureName with the Slot qualifier to filter the Property that you want to access

Note: The qualified name of that Property is the name of the qualified Stereotype plus 2 colons and the name of the Property.
Stereotype: SysML::Requirements::Requirement
Property: SysML::Requirements::Requirement::Text

4. Complete the rest of the configuration as usual: worksheet-templates, synctable and workbook.

Figure 2.3: featureName Example
2.3 stereotypeNames

Description

Use the stereotypeNames virtual feature to filter and create Model Elements with the specific combination of Stereotypes. To use this virtual feature you need a complete and necessary list of Stereotypes and their qualified names, and concatenate those qualified names into a single String. Only Elements which Stereotypes match in number and in qualified name are accepted by this filtering. The order of those Stereotypes is not important.

Syntax

This virtual feature is used as a attribute filter inside the qualifier and it does not require a dot or slash. The stereotypeNames virtual feature must be followed by an equal symbol and a String with Stereotypes. That String must separate the Stereotypes qualified names with a comma to work properly.

alias::stereotypeNames="one::qualified::name,another::qualified::name"

It is important to know that the order of the qualified names are not important. They can be swapped and the same result is to be expected. On the other hand, the String must include the exact number of Stereotypes the filter should use. Meaning if you have a model with N Elements with Stereotypes A and B, filtering using the String "A;B;C" would not show any of those N Elements as they do not have the same number of Stereotypes.

Using the stereotypeNames Virtual Feature

The following steps illustrate what you need to use the stereotypeNames virtual feature:

1. The MapleMBSE ecore is imported and its alias is mse.

2. A couple data-sources are used for this example with stereotypNames to filter Packages and Classes.

3. To use this feature to apply Stereotypes, you need to define a synctable-schema. **Note**: To avoid problems with MapleMBSE it is a good practice to use the same qualifier and Stereotypes filter in the data-source and the first dimension of the schema.

4. Complete the rest of the configuration as usual: worksheet-templates, synctable and workbook.
Example

The following example showcases how to use `stereotypeNames` to create and filter `Elements` with different `Stereotypes`.

```
1  import-core "http://www.omg.org/spec/UML/2.5.1"
2  import-core "http://maplebase.maplesoft.com/common/1.0" as ms
3
4  workbook {
5    worksheet ActivityTable(FunctionalTableSchema)
6      worksheet FunctionModelMatrix(functionalTableSchema, functionalTableSchema, functionalTableSchema)
7      worksheet FunctionalModelTable(functionalTableSchema)
8      worksheet PREARequirementTable(PREARequirementSchema)
9      worksheet DerivedPREARequirementTable(derivedPREARequirementSchema)
10     worksheet RequirementPREATable(derivedPREARequirementSchema, PREARequirementSchema, functionalTableSchema)
11  }
12
13  data-source Root(Model)
14  data-source pkg = Root/packagedElement[Model].name = "Model"/UAWSystem"/packagedElement[Package].name = "System Behavior"
15  data-source act = pkg.packagedElement[Activity];
16  data-source fset = pkg.packagedElement[Package].name = "PREA"/packagedElement[Class].name::stereotypeNames::"CustomStereotypes::PREA"
17  data-source areq = pkg.packagedElement[Package].name = "PREA"/packagedElement[Class].name::stereotypeNames::"CustomStereotypes::PREARequirement, SYRL::Requirements::AbstractRequirement"
18
19  table-schema FunctionalTableSchema{
20    key column /name as actName
21  }
22
23  table-schema PREARequirementSchema{
24    column /name as ReqName
25    column /appliedStereotypeInstance[InstanceSpecification]
26      /slot[slot[ms::Requirement::abstractRequirement::Id]/value[literalString]/value as ReqID]
27      /slot[slot[ms::Requirement::abstractRequirement::Text]/value[literalString]/value as ReqSpecification
28  }
29
30  ...
31
32  ...
33
34  ...
35
36
```

Figure 2.4: `stereotypeNames` Example
3 Associations

An *Association* between two *Blocks* creates cross references for two UML *Classes* with SysML *Block Stereotypes* (<<block>>) to one *Association* using two properties and also makes some cross references, like *Type* and *Association*, within those properties.

3.1 associatedProperty

Description

In MagicDraw, with a couple clicks from one block to another, all of these elements are correctly created. Similarly in MapleMBSE, the `associatedProperty` virtual feature provides the ability to connect two SysML *Blocks*, creating a bidirectional *Association* at the same hierarchical level in the diagram as the source *Block*.

When MapleMBSE queries the model, the `associatedProperty` returns the target *Block* (the *Block* that is related to a *Property* through an *Association*).

Syntax

The general syntax for using the `associatedProperty` virtual feature is as follows:

```
.alias::associatedProperty
```

Where `alias` is the alias you assigned to the MapleMBSE ecore. For more information on assigning aliases, see *Importing the MapleMBSE Ecore* (page 3).
The `associatedProperty` virtual feature must be used when querying the `Property` of a `Block`.

**Using the associatedProperty Virtual Feature**

The following example illustrates what you need to do to use `AssociatedProperty` virtual feature.

1. In line two, the `maplembse.ecore` is imported with an alias.
2. Use an `ownedAttribute[Property]` as the queried dimension.
3. Make a reference-query to a class using `mse::associatedProperty`.
Example

```
import.ecore "http://www.nomagic.com/magicdraw/UML/2.5"
import.ecore "http://mapembse.maplesoft.com/common/1.0" as mse

data-source Root[Model]
data-source structurePkg = Root/packageElement[Package]
data-source cls = structurePkg/packageElement[Class]
	synctable-schema ClassTableSchema {
    dim [Class] {
        key column /name as ClassName
    }
}

synctable-schema ClassTreeTableSchema(blocks: ClassTableSchema) {
    record dim [Class] {
        key column /name as className
    }

    dim /ownedAttribute[Property].mse::associatedProperty[Class] @ cls {
        reference-decomposition cls = cts {
            foreign-key column ClassName as referredClassName
        }
    }
}

synctable classTableSchema = ClassTableSchema<cls>
synctable classTreeTableSchema = ClassTreeTableSchema<cls>(classTableSchema)

worksheet-template ClassTable(cts: ClassTableSchema) {
    vertical table tab1 at (6, 2) = cts {
        key field ClassName : String
        key field Name4 : String
    }
}

worksheet-template ClassTreeTable(ctt: ClassTreeTableSchema) {
    vertical table tab1 at (6, 2) = ctt {
        key field ClassName1 : String
        key field referredClassName : String
    }
}

workbook{
    worksheet ClassTable(classTableSchema)
    worksheet ClassTreeTable(classTreeTableSchema)
}
```

Figure 3.1: associatedProperty Example
3.2 directedAssociatedProperty

Description

To create Associations with navigability in one direction MapleMBSE uses directedAssociatedProperty, using this virtual feature links two Classes and adds a Property to the source Block and other Property to an Association.

Based on the aggregation value we can use this virtual feature to create Association, Aggregation and Composition with direction.

Syntax

The general syntax for using the directedAssociatedProperty virtual feature is as follows:

```
.alias::directedAssociatedProperty
```

Where alias is the alias you assigned to the MapleMBSE ecore (hyperlink to above).

The directedAssociatedProperty virtual feature must be used when querying the Property of a Block.

Using the directAssociatedProperty Virtual Feature

The following example illustrates what you need to do to use directedAssociatedProperty.

1. In line two, the maplembse ecore is imported with an alias.
2. Use an ownedAttribute[Property] as the queried dimension.
3. Make a reference-query to a class using mse::directedAssociatedProperty.
Example

```java
import ecore "http://www.nomagic.com/magicdraw/UML/2.5"
import ecore "http://maplembse.maplesoft.com/common/1.0" as mse

data-source Root[Model]
data-source structurePkg = Root/packagedElement[Package]
data-source cls = structurePkg/packagedElement[Class]

syntax-table-schema ClassTableSchema {
  dim [Class] {
    key column /name as className
  }
}

syntax-table-schema ClassTreeTableSchema(blocks: ClassTableSchema) {
  record dim [Class] {
    key column /name as className1
  }
  dim /ownedAttribute[Property].mse::directedAssociatedProperty[Class] @ cls {
    reference-decomposition cls = cts {
      foreign-key column ClassName as referredClassName
    }
  }
}

syntaxtable classTableSchema = ClassTableSchema<cls>
syntaxtable classTreeTableSchema = ClassTreeTableSchema<cls>(classTableSchema)

worksheet-template ClassTable(cts: ClassTableSchema) {
  vertical table tab1 at (6, 2) = cts {
    key field ClassName : String
    key field Name4 : String
  }
}

worksheet-template ClassTreeTable(ctt: ClassTreeTableSchema) {
  vertical table tab1 at (6, 2) = ctt {
    key field ClassName1 : String
    key field referredClassName : String
  }
}

workbook{
  worksheet ClassTable(classTableSchema)
  worksheet ClassTreeTable(classTreeTableSchema)
}
```

Figure 3.2: directedAssociatedProperty Example
3.3 otherAssociatedEnd

Description

otherAssociationEnd is used in the case when two classifiers has to be linked and the information about the properties of these classifiers are owned by the association and not the classifiers themselves, such as in the case of UseCase diagram where association exist between an actor and usecase and these two classifiers does not own any property that defines the other classifier.

Syntax

The general syntax for using the otherAssociationEnd virtual feature is as follows:

```
.alias::otherAssociationEnd
```

Where alias is the alias you assigned to the MapleMBSE ecore (hyperlink to above).

The otherAssociationEnd virtual feature must always be used when querying a Class.

Using the otherAssociatedEnd Virtual Feature

The following example illustrates what you need to do to use otherAssociationEnd.

1. In line two, the maplembse ecore is imported with an alias.
2. Use when a Class as the queried dimension.
3. Make a reference-query to a class using mse::otherAssociationEnd, unlike other virtual features in this section otherAssociationEnd should not be used when a property is queried.
Example
Figure 3.3: otherAssociatedEnd Example

```plaintext
import-eclor "http://www.nomagic.com/magicdraw/UML/2.5"
import-eclor "http://maplembse.maplesoft.com/common/1.0" as mse

data-source Root[Model]
data-source useCasePkg = Root/packagedElement[Package]
data-source actors = useCasePkg/packagedElement[Actor]
data-source useCases = useCasePkg/packagedElement[UseCase]

synctable-schema ActorsTable {  
    record dim [Actor] {  
        key column /name as Actor  
    }  
}

synctable-schema UseCasesTable(ac:ActorsTable) {  
    record dim [UseCase] {  
        key column /name as Name  
        reference-query mse::otherAssociationEnd[Actor] @ actor  
        reference-decomposition actor = ac {  
            foreign-key column Actor as Actor  
        }  
    }  
}

synctable actorsTable = ActorsTable<actors>
synctable useCasesTable = UseCasesTable<useCases>(actorsTable)

worksheet-template Actors(ac: ActorsTable) {  
    vertical table tab1 at (5, 3) = ac {  
        key field Actor : String  
    }  
}

worksheet-template UseCases(auct: AssociatedUseCasesTable) {  
    vertical table tab1 at (5, 3) = auct {  
        key field Name : String  
        key field Actor : String  
    }  
}

workbook {  
    worksheet Actors(actorsTable)  
    worksheet UseCases(useCasesTable)  
}
3.4 nestedDirectedComposition

Description

MapleMBSE is powerful enough to change any SysML feature, in particular a `nestedClassifier`. The effort to change the model in a desired way is always related to creating the right schemas and data sources to offer intuitive views. Unfortunately, creating a nested block and a directed composition to it is not an easy task without this virtual feature. The creation of composition association to non-existing nested block should be possible just by mentioning the name of the target block in the right dimension.

Syntax

The general syntax for using the `nestedDirectedComposition` virtual feature is as follows:

```
dim /alias::nestedDirectedComposition[Association]
```

Where `alias` is the alias you assigned to the MapleMBSE ecore (hyperlink to above).

The `nestedDirectedComposition` virtual feature must be used when querying the `Block` and `Association` in a dimension. It always needs to be used in conjunction with another virtual feature to set up the target `Block: targetBlockName`.

The general syntax for using the `targetBlockName` virtual feature is as follows:

```
key column /alias::targetBlockName
```

This `targetBlockName` virtual feature should be the only key column for a `nestedDirectedComposition` dimension.

Using the `nestedDirectedComposition virtual feature`

The following example illustrates what you need to do to use `nestedDirectedComposition`

1. In line two, the `maplembse.ecore` is imported with an alias.
2. Use a `Class` or an `Association` qualifier in the queried dimension, as shown in line 8, 12 and 16.
3. Create a `targetBlockName` key column for each `nestedDirectedComposition dimension`
Example

```plaintext
1  data-source Root[Model]
2
3  data-source blocks = Root
4  /packagedElement[Package|name="Structure"]
5  /packagedElement[Class|mse::metaclassName="SysML::Blocks::Block"]

6  synctable-schema Schema {
7      record dim [Class|mse::metaclassName="SysML::Blocks::Block"] {
8          key column /name as bName
9      }
10
11      record dim /mse::nestedDirectedComposition[Association] {
12          key column /mse::targetBlockName as nbName
13      }
14
15      record dim /mse::nestedDirectedComposition[Association] {
16          key column /mse::targetBlockName as nbName2
17      }
18
19  }
20
21  worksheet-template Template(sc: Schema){
22      vertical table tbl at (4,2) = sc {
23          key field bName
24          key field nbName
25          key field nbName2
26          sort-keys bName, nbName, nbName2
27      }
28
29  }
30
31  synctable blockTable = BlockSchema<rBlocks>
32  synctable dataTable = Schema<blocks>
33  synctable associatedDataTable = AssociatedPropertySchema<blocks>(blockTable)
34
35  workbook {
36      worksheet Template(dataTable) {label="Nested Classifier"}
37  }
```
4 Blocks

4.1 recursivePartProperties

Description

The recursivePartProperties virtual feature helps find all the related blocks and sub-blocks and part properties, recursively.

Displaying a block and related sub-blocks in the same syncview is difficult if they are in different packages, and there is a chance that relevant blocks are missing in the syncview. The recursivePartProperties virtual feature helps find all the related blocks, sub-blocks and part properties, recursively. This makes it easier to create a corresponding configuration file.

The recursivePartProperties virtual feature works in a similar fashion to recursiveInstanceWithSlots, and a common use case is to use both of these in conjunction for instance matrices.

Syntax

The configuration file syntax for using recursivePartProperties is illustrated below.

```plaintext
datasource blocks = Root/packageElement[Package|name="Test"]/packageElement[Class|name="B1"]/mse::recursivePartProperties[Class];

synctable-schema TestSchema {
  record dim[Class]{
    key column /name as mainBlock
  }
  Dim /mse::recursivePartProperties[Class]{
    Key column /name as subBlocks
  }
}
```

Using the recursivePartProperties Virtual Feature

The following example illustrates one way to use the recursivePartProperties virtual feature:

1. Import the MapleMBSE.ecore with an alias.
2. Create a datasource that has the context/main block for which you want to find the properties (for example, ../packageElement[Class|name="B1"]).
3. Use `recursivePartProperties[Class]` to return all the classes linked to the context/main block and sub blocks.
4. Create a sync-schema and synctable and after that use that datasource in the synctable.

**Example**

datasource blocks = Root/packageElement[Package|name="Test"]/packageElement[Class|name="B1"]/mse::recursivePartProperties[Class];

4.2 propertyDefaultValue

**Description**

Previously, to view or edit the default value of the value property or property without any Stereotype, the author/editor of the configuration file editor had to write the line column /value[LiteralReal]/value. If the property contains a value other than a real value then MapleMBSE will not display this value in the cell. If the configuration editor were to write the MSE file in such a way as to view the value of every type of value property, It will complicate the MSE file and still, the end user will not able to view the values in one single column. The propertyDefaultValue virtual feature fixes this problem and helps the configuration file editor and the end user to view and edit the value in one column.

**Syntax**

/mse::propertyDefaultValue

**Using the propertyDefaultValue Virtual Feature**

1. Import the MapleMBSE.ecore with an alias.
2. Create a datasource that has the blocks/classes for which you want to find the properties(for example, ../packageElement[Class]).
3. Use ownedAttribute[Property] to get properties from the block/Class
4. Use the propertyDefaultValue to get the value from the property
5. Create a sync-schema and synctable and then use that datasource in the synctable.

**Example**

```plaintext
synctable-schema Schema{
    record dim [Class|mse::metaclassName="SysML::Blocks::Block"] {
        key column /name as BlockName
    }
    record dim /ownedAttribute[Property mse::metaclassName |="MD Customization for SysML::additional_stereotypes::ValueProperty"] {
```
4.3 getAllProperties

Description

The `getAllProperties` virtual feature retrieves properties under a block. These properties can be direct or indirect. If the block has generalization it can go as much as possible in the upper direct to get the properties but for the composition it can only go one step down.

The images below show the model, MSE file, and the view in MapleMBSE. In the model block, Comp11, and Comp2 are generalized to GenBlock using this feature MapleMBSE can query and modify the properties that are also inherited from the GenBlock.
Note: This feature can be used with the `recursivePartProperties` to get the flat view or can be used alone for the hierarchical view.
Syntax

dim /mse::getAllProperties

Using the getAllProperties virtual feature

1. Import the MapleMBSE.ecore with an alias.
2. Create a datasource that has the blocks/classes for which you want to find the properties (for example, ../packageElement[Class]).
3. Use mse::getAllProperties[Property] to get properties from the block/Class (directly owned properties or inherit properties).
4. Create a sync-schema and synctable and after that use that datasource in the synctable.
Example

```html
<html>
    <head></head>
    <body>
        <pre>
            <code>
xncatable-schema BlocksValueTable {
            dim[Class[mse::metaclassName="SysML::Blocks::Block"]{
                key column /name as BlockName
            }
            dim /mse::getAllProperties[Property
            [mse::metaclassName="MD Customization for SysML::additional_stereotypes::ValueProperty",
                aggregation="composite"] {
                key column /name as ValueProp
                column /mse::propertyDefaultValue as defaultValue
            }
        }
    </pre>
</body>
</html>
```
5 Connectors

A Connector is used to link ConnectableElements (for example, Ports or Properties) of a Class through a ConnectorEnd. A Connector has two ConnectorEnds.

Based on the connection between Properties of a Class the connection can be of two types: Delegation (connecting Ports or Properties from the system to Ports or Properties inside a Class) or Assembly (connecting Ports or Properties within a Class).

5.1 connectedPropertyOrPort

Description

To achieve this connection MapleMBSE uses connectedPropertyOrPort virtual feature.

The connectedPropertyOrPort virtual feature connects Ports or Properties of a Class. It automatically detects the kind of relation required between the Properties being connected and creates the appropriate connection.

When MapleMBSE queries the model, the connectedPropertyOrPort return the list of target properties.

Syntax

The general syntax for using the connectedPropertyOrPort virtual feature is as follows:

```
.alias::connectedPropertyOrPort
```

Where the alias is alias you assigned to MapleMBSE ecore.
When the connection is created through `connectedPropertyOrPort`, the owner of the connected `Property` is determined automatically by MapleMBSE, regardless of whether this is a `Delegation` or `Assembly` type connection.

**Using the connectedPropertyOrPort virtual feature**

In general, to use the `connectedPropertyOrPort` virtual feature:

1. First, import the MapleMBSE ecore with alias
2. Use an `ownedAttribute[Property]` as the queried dimension.
3. Make a reference-query to a property using `mse::connectedPropertyOrPort`.

**Example**

A specific example of how to use the `ConnectedPropertyOrPort` virtual feature is shown below.

```
1  import-ecore "http://www.nomagic.com/magicdraw/UML/2.5"
2  import-ecore "http://maplembse.maplesoft.com/common/1.0" as mse
3
4    syncable-schema BlocksTable {
5      record dim [Class| mse::metaclassName="SysML::Blocks::Block"] { 
6        key column /name as BlockName 
7      } 
8      dim /ownedAttribute[Property] { 
9        key column /name as PropertyName 
10     } 
11  }
12
13    syncable-schema ConnectedPropertyOrPortTable(bT: BlocksTable) { 
14      record dim [Class| mse::metaclassName="SysML::Blocks::Block"] { 
15        key column /name as className 
16      } 
17      record dim /ownedAttribute[Property] { 
18        key column /name as ParentPort 
19      } 
20      record dim .mse::connectedPropertyOrPort @ cls { 
21        reference-decomposition cls = bT { 
22          foreign-key column BlockName as referredClassName 
23          foreign-key column PortName as referredPortName 
24      } 
25  }
```

**Figure 5.1: connectedPropertyOrPort Example**
5.2 otherConnectorEnd

Description

To achieve this connection MapleMBSE also use otherConnectorEnd virtual feature. This virtual feature can connect between ports or properties of a class, otherConnectorEnd automatically create the relation required between the properties being connected and creates appropriate connection.

When MapleMBSE queries the model, the otherConnectorEnd return the list of connectorEnds which is associated with the property.

Syntax

The general syntax for using the otherConnectorEnd virtual feature is as follows:

```
.alias::otherConnectorEnd
```

Where the alias is the alias you assigned to the MapleMBSE ecore.

When the connection is created using otherConnectorEnd, the owner of the connected Property is determined automatically by MapleMBSE, regardless of whether this is a Delegation or Assembly type connection.

Using the otherConnectorEnd Virtual Feature

How to use the otherConnectorEnd virtual feature is shown in the example below:

1. First, import the MapleMBSE ecore with an appropriate alias
2. Use an ownedAttribute[Property] as the queried dimension.
3. Make a reference-query to a property using mse::otherConnectorEnd.

Example

A specific example of how to use the otherConnectorEnd virtual feature is shown below.
Figure 5.2: otherConnectorEnd Example
6 Dependencies

A Dependency is used between two model elements to represent a relationship where a change in one element (the supplier element) results in a change to the other element (client element).

A Dependency relation can be created between any namedElement. Different kinds of Dependencies can be created between the model elements such as Refine, Realization, Trace, Abstraction etc.,

6.1 clientDependencies

Description

The clientDependencies virtual feature creates a relation between the client being the dependent and supplier who provides further definition for the dependent.

Syntax

The general syntax for using the clientDependencies virtual feature is as follows:

/mse::clientDependencies

This virtual feature is used while querying a Class that has to be assigned as client to the dependency that is being created and is used in a following dimension the class that is being queried.

Where alias is the alias you assigned to the MapleMBSE ecore.

Using the clientDependencies Virtual Feature

In general, the following steps outline how to use clientDependencies:

1. It should be used when a named element is queried
2. Information about the type of relationship is specified as [Dependency], [Abstraction] etc.,
3. When querying the model element with mse::clientDependencies, the reference decomposition should be to a supplier element.

Example

The example below is an illustration of how to use the clientDependencies virtual feature.
6.2 supplierDependencies

Description
Similar to clientDependencies, supplierDependencies is used to create a relation between two named elements. The only difference between the two virtual features is supplierDependencies is used when the relationship has to be made from supplier to client instead of client to supplier, as in the case of clientDependencies.

**Syntax**

The general syntax for using the supplierDependencies virtual feature is as follows:

```
/mse::supplierDependencies
```

This virtual feature is used while querying a Class that has to be assigned as supplier to the dependency that is being created and is used in a dimension following the class that is being queried.

Where alias is the alias you assigned to the MapleMBSE.ecore.

**Using the supplierDependencies Virtual Feature**

The following example illustrates what you need to do to use supplierDependencies

1. It should be used when a named element is being queried.
2. Information about the type of relationship is specified as [Dependency], [Abstraction] etc.,
3. When querying the model element with mse::supplierDependencies the reference decomposition should be to a client element.
Example

```java
import-core "http://www.nomagic.com/magicdraw/UML/2.3"
import-core "http://maplembse.maplesoft.com/common/1.0" as msc

data-source Root(Model)
data-source package = Root/package/Package

data-source cls = package/package/Class

// table schema
sync-table schema RequirementsTableSchema { record dim [class msc::metaclassName="SysML::Requirements::Requirement"] {
    key column /name as ReqName
}

// table schema
sync-table schema RequirementsDerivesTableSchema (rule RequirementsTable) { record dim [class msc::metaclassName="SysML::Requirements::Requirement"] {
    key column /name as ReqName
}

// table schema
sync-table schema RequirementsDependencies (cls:RequirementsTable) { record dim /msc::supplierDependencies[Abstraction] {
    key reference-query .client @ reqDecomp
    reference-decomposition reqDecomp = req {
        foreign-key column ReqName as DeriveName
    }
}

// worksheet
worksheet-template ReqClassTable (cts:RequirementsTableSchema) {
    vertical table tabi at (4,5) = cts{
        key field Name : String
}

// worksheet
worksheet-template ReqClassDependency (cds:RequirementsDerivesTableSchema) {
    vertical table tabi at (4,5) = cds{
        key field DeriveName : String
}

// workbook
workbook{
    worksheet ReqClassTable (requirementsTableSchema)
    worksheet ReqClassDependency (requirementsDerivesTableSchema)
}
```

Figure 6.2: supplierDependencies Example
7 Enumeration

Enumeration is a special DataType that can be compared to a list of possible values, the way that "colors" can be an enumeration and possible values can be: red, blue, green, etc. These Enumerations are composed of EnumerationLiterals which are the different values and the actual Elements to be referenced. MapleMBSE supports a couple virtual features that need to be used in conjunction to access and reference any Enumeration and its EnumerationLiterals independently of where in the TWCloud project those values are stored (for example, under Model or customized profile)

7.1 EnumerationName

Description

MapleMBSE, to simplify Enumeration identification, supports an enumerationName virtual feature that allows simpler access to a specific Enumeration while creating an MSE configuration. Note that MapleMBSE, while using this virtual feature, will by default instantiate the accessed Element to the first EnumerationLiteral of the Enumeration. Nonetheless, enumerationLabel can be used to change to another EnumerationLiteral. See the next section for further details.

Syntax

The general syntax for using the enumerationName virtual feature is as follows:

alias::enumerationName="qualified::name"

Where alias is the alias you assigned to the MapleMBSE ecore and qualified::name is the qualifiedName of the Enumeration. For more information on assigning aliases, see Importing the MapleMBSE Ecore (page 3).

The enumerationName virtual feature must be used while querying an Element with a Stereotype that supports some Property with an Enumeration type. For more information on how to access a Slot, see the sections in the guide on the metaclassName and featureName virtualFeatures. Once you get the specific Slot, retrieve its value and within its Qualifier filter use enumerationName.

Using the enumerationName Virtual Feature

The following example illustrates what you need to do to use the enumerationName virtual feature:

1. Import the maplembse ecore with an alias.

2. Create a schema that takes an Element with a Stereotype and navigate down to its InstanceValue for a Property with an Enumeration type. See lines 15 to 18 in the example code in the next section for an illustration.
3. Make sure you are using the right combination of qualified names for Stereotypes, Slot Properties and Enumeration.

4. Complete the /value[InstanceValue] navigation with an enumerationLabel (see next section for further details).

Example

```plaintext
import.ecore "http://www.nomagic.com/magicdraw/SPL/2.5"
import.ecore "http://maplembse.maplesoft.com/common/1.0" as mbe

workbook {
  worksheet EnumerationTemplate(enums)
}

data-source Root[Model]

data-source reqs : Root[packagedElement][Package][name="Enum"]
  /packagedElement[Class][mbe:metaclassName="SysML:Non-Normative Extensions::Requirement::extendedRequirement"]

syntable-schema EnumSchema {
  
  die [Class][mbe:metaclassName="SysML:Non-Normative Extensions::Requirement::extendedRequirement"] {
    
    key column /name as name
    
    column[appliedStereotypeInstance][InstanceSpecification]
      /Slot[Slot][mbe:featureName="SysML:Non-Normative Extensions::Requirement::extendedRequirement::verifyMethod"]
        /value[InstanceValue][mbe:enumerationName="SysML:Non-Normative Extensions::Requirement::VerificationMethodKind"]
  
  /mbe:enumerationLabel as verificationMethod
  
}

syntable enums = EnumSchema(reqs)

worksheet template EnumerationTemplate (es: EnumSchema) {
  
  vertical table tab1 at (1, 1) - es {
    
    key field rName
    
    field verificationMethod
  }
}
```

7.2 EnumerationLabel

Description

As shown in the previous sections on EnumerationName, MapleMBSE allows you to make a reference to Enumeration using a qualifiedName. However, without the right mechanism to translate from String to EnumerationLiterals and vice versa, the end user will be forced to deal with strange Object references or unusable Excel cells. This is exactly the problem enumerationLabel was designed to solve. Using this virtual feature allows the end user to see the String name of the EnumerationLiteral without forcing any reference-decomposition and it allows also the end user to change the reference from the Slot Property using the String name of the desired EnumerationLiteral.

Syntax

The general syntax for using the enumerationLabel virtual feature is as follows:

```
(alias::enumerationLabel)
```

Where alias is the alias you assigned to the MapleMBSE.ecore. For more information on assigning aliases, see Importing the MapleMBSE Ecore (page 3).

The enumerationLabel virtual feature must be used while querying an InstanceValue with a Stereotype that supports some Property with a Enumeration type and which was
filtered with `enumerationName`. For more information how to access this kind of `InstanceValue`, see the previous section.
8 Util

This section contains all other virtual features that do not create elements but offer a better alternative to access and map model information.

8.1 multiplicityProperty

Description

The UML specification contains several MultiplicityElements like Properties that have upper and lower features to describe their multiplicity. Use the multiplicityProperty virtual feature to make a configuration that translates a string into those upper and lower values and the other way around.

This virtual feature recognizes the UML commonly used notation for multiplicity (e.g. 0..*). Supporting this notation makes MapleMBSE much easier to use without adding complexity and thus the final user has less to input into Excel.

Syntax

The general syntax for using the multiplicityProperty virtual feature is as follows:

/alias::multiplicityProperty

Where the alias is the alias you assigned to the MapleMBSE.ecore.

This virtual feature can only be used while querying a concrete EClass implementing a MultiplicityElement like a Property or a Pin. A slash notation is needed prior to the alias, the 2 colons, and multiplicityProperty.

As mention previously multiplicityProperty uses a string to represent the multiplicity, meaning that this particular virtual feature cannot being used as a dimension with a qualifier. It is intended to be used only at a column declaration.

Using the multiplicityProperty Virtual Feature

The following example shows you how to map the multiplicity of a concrete MultiplicityElement like Property and a string.

1. Import the MapleMBSE.ecore, as usual the alias used is mse
2. Inside a syntable-schema navigate to a MultiplicityElement, in this case /ownedAttribute[Property] within a Class
3. Within that dimension, define a regular column using /mse::multiplicityProperty
4. Complete the rest of the configuration as usual: \texttt{worksheet-templates}, \texttt{synctable} and \texttt{workbook}

\textbf{Example}

```java
import.ecore "http://www.nomagic.com/magicdraw/UML/2.5"
import.ecore "http://maplembse.maplesoft.com/common/1.0" as mso

data-source Root[Model]
data-source classes = Root/packagedElement[Class]

synctable-schema Schema {
  record dim [Class] {
    key column /name as cName
  }

  record dim /ownedAttribute[Property] {
    key column /name as pName
    column /mso::multiplicityProperty as multiplicity
  }
}

worksheet-template Template(sch: Schema) {
  vertical table tab1 at (2, 2) = sch {
    key field cName : String
    key field pName : String
    field multiplicity : String
    sort-keys cName, pName
  }
}

synctable tableProperty = Schema<classes>

workbook {
  worksheet Template(tableProperty)
}
```

Figure 8.1: \texttt{multiplicityProperty} Example
9 Activity Diagrams

An Activity Diagram is a diagram with a direct connection, ActivityEdge, that connects a node, ActivityNode, to another ActivityNode. An Activity Diagram is useful to abstract behavioral information within a system. In order to improve MSE configurations, MapleMBSE supports control and object flow, the 2 kind of ActivityEdges, with two distinct virtual features.

9.1 ActivityControlFlow

Description

A ControlFlow is an ActivityEdge that is used to control the execution of ActivityNodes within an Activity.

In MapleMBSE, the virtual feature ActivityNode is used as a reference to create ControlFlows. Note that in MapleMBSE, abstract classes such as ActivityNode cannot be instantiated. Thus, you must instantiate concrete classes such as CallActionBehavior, ActivityParameterNode, or InitialNode. See the example section for further details.

Syntax

The general syntax for using the activityControlFlow virtual feature is as follows:

```
.alias::activityControlFlow
```

Where alias is the alias you assigned to the MapleMBSE ecore. For more information on assigning aliases, see Importing the MapleMBSE Ecore (page 3).

The activityControlFlow virtual feature must be used when querying the ActivityNode of Activity.

Using the ActivityControlFlow Virtual Feature

The following example illustrates what you need to do to use activityControlFlow virtual feature:

1. Import the maplembse.ecore with an alias.
2. Create a schema that navigates until an ActivityNode or an element that has an ActivityNode as its first dimension
3. Make a dimension reference-query to another ActivityNode using .mse::activityControlFlow.

This example has extra schema, CallBehaviorActionSchema used to create concrete ActivityNodes. The other schemas in this example will fail to instantiate Element because
ActivityNode is an abstract class.

**Note:** Some data sources specific to a fictional project were created to simplify the reference-decomposition. In a real life scenario you might need to identify the Package, the Activity and the ActivityNode that you want to connect to.

**Example**

```java
import ecore "http://www.nomagic.com/magicdraw/UML/2.5"
import ecore "http://maplembse.maplesoft.com/common/1.0" as mse

data-source Root[Model]
data-source pkg = Root/package+element[Package|name = "controlflow"]
data-source activities = pkg/package+element[Activity|name="activity"]
data-source nodes = activities/node[ActivityNode]
data-source cba = activities/node[CallBehaviorAction]

tsnc.xsd NodeSchema {
  dim [ActivityNode] {
    key column /name as nName
  }
}

tsnc.xsd CallBehaviorActionSchema {
  dim [CallBehaviorAction] {
    key column /name as nName
  }
}

worksheet-template CallBehaviorActionTemplate (cbasc: CallBehaviorActionSchema) {
  vertical table tab1 at (2, 1) - cbasc {
    key field nName
  }
}

tsnc.xsd Schema(nsc: NodeSchema) {
  dim [ActivityNode] {
    key column /name as nName
  }

  dim .mse::activityControlFlow[ActivityNode] @ tgtNode {
    reference-decomposition tgtNode = nsc {
      foreign-key column nName as tgtNode
    }
  }
}
```

**Figure 9.1: ActivityControlFlow Example**
9.2 ActivityObjectFlow

Description

An ObjectFlow is an ActivityEdge that represents the flow of object data between ActivityNodes within an Activity. Sometimes, the ObjectFlow directly connects two ActivityNodes. However, due to UML specifications, some ActivityNodes cannot be connected directly using an ObjectFlow. In these cases Pins are required. Pins are objects that accept and provide values to actions. These values represent an input to an action or output from an action.

If an ActivityNode that requires Pins, such as CallBehaviorAction, also has a Behavior that further describes it’s functionality, then both the ActivityNode and Behavior need to have their Pins (specifically ActivityParameterNode and Parameters) synchronized, both in quantity and direction.

Syntax

The general syntax for using the activityObjectFlow virtual feature is as follows:

```
.alias:: activityObjectFlow
```

Where alias is the alias you assigned to the MapleMBSE ecore. For more information on assigning aliases, see Introduction (page 1).

The activityObjectFlow virtual feature must be used when querying the ActivityNode of Activity.

Using the ActivityObjectFlow Virtual Feature

The following example illustrates what you need to do to use activityObjectFlow virtual feature:

1. Import the maplembse ecore with an alias.
2. Create a schema that navigates until an ActivityNode or an element which has an ActivityNode as its dimension.
Example

```java
import-eore "http://www.nomagic.com/magicdraw/UML/2.5"
import-eore "http://maplembse.maplesoft.com/common/1.0" as mse

data-source Root[Model]
data-source pkg = Root/packagedElement[Package|name = "objectflow"]
data-source activities = pkg/packagedElement[Activity|name = "activity"]
data-source nodes = activities/node[ActivityNode]

syncatable-schema NodeSchema {
  dim [ActivityNode] {
    key column /name as nName
  }
}

syncatable-schema Schema(nsc: NodeSchema) {
  dim [ActivityNode] {
    key column /name as nName
  }

  dim .mse::activityObjectFlow[ActivityNode] @ tgtNode {
    reference-decomposition tgtNode = nsc {
      foreign-key column nName as tgtNode
    }
  }
}

worksheet-template Template (sc: Schema) {
  vertical table tab1 at (2, 1) = sc {
    key field nName
    key field tgtNode
  }
}

syncatable nodeTable = NodeSchema<nodes>
syncatable controlFlowTable = Schema<nodes>(nodeTable)

workbook {
  worksheet Template(controlFlowTable)
}
```

Figure 9.2: ActivityObjectFlow Example
10 StateMachines

StateMachine diagrams are used to define the different states that a system will exist in. This kind of diagram helps modelers to describe discrete, event-driven behaviors of the whole system or its parts.

10.1 VertexTransition

Description

MapleMBSE, in order to simplify Transition between Vertices, supports a vertexTransition virtual feature that allows a better end user experience while inputting data. Note that MapleMBSE will fail to instantiate abstract classes like Vertex and it will be required to instantiate instead concrete classes like Pseudostate, State or FinalState. Nonetheless, Vertex can be used as reference to create Transitions. See the example section for further details.

Syntax

The general syntax for using the vertexTransition virtual feature is as follows:

`.alias:: vertexTransition`

Where alias is the alias you assigned to the MapleMBSE ecore. For more information on assigning aliases, see Importing the MapleMBSE Ecore (page 3). The vertexTransition virtual feature must be used when querying the any kind of Vertex within a given Region of a StateMachine.

Using the VertexTransition Virtual Feature

The following example illustrates what you need to do to use the vertexTransition virtual feature:

1. Import the maplembse ecore with an alias.
2. Create a schema that navigates till an Vertex or which first dimension is an Vertex.
3. Make a dimension reference-query to another Vertex using .mse:: vertexTransition.

This example has some extra schema, called StateSchema, used to create concrete States. The other schemas in this example will fail to instantiate Element because Vertex is an abstract class.

Note: some data sources specific to a fictional project were create in order to simplify the reference-decomposition, in a real life scenario you might need to identify the Package, the StateMachine, the Region and the Vertex that you want to connect to.
Example

```java
import-ecl "http://www.nomagic.com/magicdraw/UML/2.5"
import-ecl "http://maplembc.maplesoft.com/ecoem/1.9" as mse

data-source Root[Model]
data-source pkg = Root/packageElement[Package[name="statemachine"]]
data-source stm = pkg/packageElement[StateMachine[name="stm"]]
data-source rg = stm/region[Region[name="rg"]]
data-source vertices = rg/subvertex[Vertex]
data-source states = rg/subvertex[State]

@syntaxtable-schema VertexSchema {
  dim [Vertex] {
    key column /name as vName
  }
}

@syntaxtable-schema StateSchema {
  dim [State] {
    key column /name as vName
  }
}

worksheet-template StateTemplate(ssc: StateSchema) {
  vertical table tab1 at (2, 1) = ssc {
    key field vName
  }
}

syntaxtable-schema Schema(vsc: VertexSchema) {
  dim [Vertex] {
    key column /name as vName
  }
  dim mse::vertexTransition[Vertex] @ tgtRef {
    reference-decomposition tgtRef = vsc {
      foreign key column vName as tgtVertex
    }
  }
}

worksheet-template Template(ssc: Schema) {
  vertical table tab1 at (2, 1) = ssc {
    key field vName
    key field tgtVertex
  }
}

syntaxtable vertexTable = VertexSchema<vertices>
syntaxtable stateTable = StateSchema<states>
syntaxtable transitionTable = Schema<vertices>(vertexTable)

worksheet {
  worksheet StateTemplate(stateTable)
  worksheet Template(transitionTable)
}
```

Figure 10.1: VertexTransition Example
11 Comments

11.1 ownedComments

Description

A comment is an element that represents a textual annotation that can be attached to other elements or a set of elements.
A comment can be owned by any element.
This virtual feature creates a comment and annotated it with the owner element.

Syntax

The general syntax for using the ownedComments virtual feature is as follows:

/alias:: ownedComments

Where the alias is the alias you assigned to the MapleMBSE ecore.

Using the ownedComments Virtual Feature

The following example shows you how to use the ownedComments feature.

1. Import the MapleMBSE ecore, as usual the alias used is mse
2. Inside a syntable-schema navigate to a Class
3. Within that dimension, define a regular column using /mse::ownedComments[Comment]
4. Complete the rest of the configuration as usual: worksheet-templates, syntable and workbook
Example

```java
import.ecore "http://www.nomagic.com/magicdraw/UML/2.5"
import.ecore "http://maplembse.maplesoft.com/common/1.0" as mse

data-source Root[Model]
data-source blocks = Root/packagedElement[Package]

syntax-schema Schema {
  record dim [Package] {
    key column /name as packageName
  }

  record dim /packagedElement[Class] {
    key column /name as className
    column /mse::ownedComments[Comment]/body as body
  }
}

worksheet-template Template(ts: Schema) {
  vertical table t at (1,1) = ts {
    key field packageName
    field className
    field body :String
    sort-keys packageName
  }
}

syntax-schema Schema2 {
  record dim [Package] {
    key column /name as packageName
  }

  record dim /packagedElement[Class] {
    key column /name as className
  }

  record dim /mse::ownedComments[Comment] {
    key column /body as body
  }
}

worksheet-template Template2(ts: Schema2) {
  vertical table t at (1,1) = ts {
    key field packageName
    field className
    field body :String
    sort-keys packageName
  }
}

syntax Table1 = Schema<blocks>
syntax Table2 = Schema2<blocks>

workbook {
  worksheet Template(Table1)
  worksheet Template2(Table2)
```
12 Slots

SysML permits users to create an instance of the classifiers with their properties. If the classifier is defined with some properties, the instances will own slots that contain the properties defined. This instance allows users to create concrete elements from the more general model. In order to simplify the task related to InstanceSpecification, MapleMBSE proposes the following virtual features to support the creation, edition, and removal of instances and their Slots.

12.1 SlotValue

Description

SysML has a complex structure to access the values within Slots. Those values change widely depending on the type of the property defining the owning Slot. Imagine a real property defining Slot, which, in order to contain that value, requires a LiteralReal, and then the real value will be stored within that literal. Each type has its own literal class, and for reference to other instances the mechanisms are another matter altogether. This is just a reminder of how much complexity this InstanceSpecification modeling has, but thanks to this virtual feature, MapleMBSE simplifies and hides that complexity. Using a single access point and without caring about the concrete type of the property, SlotValue will return a string representing the value given Slot. MapleMBSE proposes an easy mechanism to display, create, and edit that first value associated to any Slot. Emphasis in first, SysML metamodel allows to associate several values to a single Slot, it is by design that MapleMBSE does not use this virtual feature for a different multiplicity.

Syntax

The general syntax for using the slotValue virtual feature is as follows:

```
column [Slot]/alias::slotValue as column_name
```

Where alias is the alias you assigned to the MapleMBSE ecore. For more information on assigning aliases, see Importing the MapleMBSE Ecore (page 3). The slotValue virtual feature must be used while querying a Slot, and the return string can only be used within a column.
Using the SlotValue Virtual Feature

The following example illustrates what you need to do to use the slotValue virtual feature:

1. Import the MapleMBSE.ecore with an alias
2. Create a schema that has a dimension accessing a Slot from an InstanceSpecification, see line 24.
3. Make sure that you are using the right combination of applied Classifier to the InstanceSpecification and the Slot’s definingFeature.
4. Access that Slot’s value using the slotValue virtual feature, see line 30
Example

```
import.ecore "http://www.nomagic.com/magicdraw/UML/2.5.1"
import.ecore "http://maplembse.maplesoft.com/common/1.0" as mse

data-source Root[Model]
data-source pkg = Root/packagedElement[Package[name="Slots"]]
data-source blocks = pkg/packagedElement[Class[mse::metaclassName="SysML::Blocks::Block"]]
data-source instances = pkg/packagedElement[ InstanceSpecification]

syntax-schema PropertySchema {
  record dim [Class[mse::metaclassName="SysML::Blocks::Block"]] {
    key column /name as bName
  }
  dim /ownedAttribute[Property] {
    key column /name as pName
  }
}
syntax-schema Schema"(psc: PropertySchema){
  record dim [InstanceSpecification] {
    key column /name as iName
  }
  dim /slot[Slot] {
    key reference-query .definingFeature @pRef
    reference-decomposition pRef = psc {
      foreign-key column bName as bName
      foreign-key column pName as pName
    }
    column /mse::slotValue as value
  }
}
worksheet-template Template(sc: Schema) {
  vertical table tab1 et"(2, 1) = sc {
    key field iName
    key field bName
    key field pName
    field value
    sort-keys iName, bName, pName
  }
}
syntaxtable propertyTable = PropertySchema<blocks>
syntaxtable syncTable = Schema<instances>(propertyTable)

workbook {
  worksheet Template(syncTable)
}
```

12.2 InstanceTree

Description

SysML forces each Slot to be owned by an InstanceSpecification. The regular way to navigate would be from InstanceSpecification to Slot, and without any other mechanisms it
would be hard to get a list of the InstanceSpecification tree for a given Slot. Remember that a Slot can have, as values, references to other InstanceSpecifications, and those would be part of the tree for that given Slot. Returning this special tree list of InstanceSpecifications is the goal of instanceTree virtual feature.

**Syntax**

The general syntax for using the instanceTree virtual feature is as follows:

```plaintext
dim .alias::instanceTree[InstanceSpecification]
```

Where alias is the alias you assigned to the MapleMBSE ecore.

For more information on assigning aliases, see *Importing the MapleMBSE Ecore (page 3)*. The instanceTree virtual feature must be used in a dimension level after querying a Slot, the return type is a list of references to the InstanceSpecifications which belong to the tree of the queried Slot.

**Using the InstanceTree Virtual Feature**

The following example illustrates what you need to do to use the instanceTree virtual feature:

1. Import the MapleMBSE ecore with an alias
2. Create a schema that has a dimension accessing a Slot from an InstanceSpecification, see line 24.
3. The dimension after the Slot one should use the instanceTree, see line 32
Example

```java
import.ecore "http://www.nomagic.com/magicdraw/UML/2.5.1"
import.ecore "http://maplemse.maplesoft.com/common/1.0" as mse

data-source Root[Model]
data-source pkg = Root/packagesElement[Package[name="Slots"]]
data-source blocks = pkg/packagesElement[Class[mse::metaclassName="SysML::Blocks::Block"]]
data-source instances = pkg/packagesElement[InstanceSpecification]

syncTable-scheme PropertySchema {
  record dim [Class[mse::metaclassName="SysML::Blocks::Block"] {
    key column /name as bname
  }

  dim /ownedAttribute[Property] {
    key column /name as pName
  }
}

syncTable-scheme Schema (psc: PropertySchema){
  record dim [InstanceSpecification] {
    key column /name as iName
  }

  record dim /slot[Slot] {
    key reference-query .definingFeature @pRef
    reference-decomposition pRef = psc {
      foreign-key column bName as bName
      foreign-key column pName as pName
    }
  }

  dim .mse::instanceTree[InstanceSpecification] {
    key column /name as iNameTree
  }
}

worksheet-template Template(sc: Schema) {
  vertical table table at (2, 1) = sc {
    key field iName
    key field bName
    key field pName
    key field iNameTree
    sort-keys iName, bName, pName
  }
}

syncTable propertyTable = PropertySchema<blocks>
syncTable syncTable = Schema<instances>(propertyTable)

workbook {
  worksheet Template(syncTable)
}
```
12.3 InstanceWithSlots

Description

It is well known that InstanceSpecifications and their Slots are an essential part of a useful and meaningful model. They are necessary to achieve results, but the task of instantiating, editing, and removing those elements is slow and error prone. MapleMBSE helps to create very complex structures using InstanceWithSlots, when you pass Class as parameter to an InstanceSpecification using this virtual feature, you will see how:

• MapleMBSE updates the list of classifiers that are applied to a given InstanceSpecification
• For each defining property related to that applied class, MapleMBSE will create a Slot defined by a property with its default value.

Syntax

To use instanceWithSlots virtual feature as a column within an InstanceSpecification dimension, the syntax is as follows:

```
reference-query .alias::instanceWithSlots @reference_name
```

This configuration line needs to be completed with a reference-decomposition that uses a Class schema, see the example for further information. Also remember that alias is the alias you assigned to the MapleMBSE ecore.

For more information on assigning aliases, see Importing the MapleMBSE Ecore (page 3).

Using the InstanceWithSlots Virtual Feature

The following example illustrates one way to use the instanceWithSlots virtual feature:

1. Import the MapleMBSE ecore with an alias
2. Create a schema that has a dimension accessing an InstanceSpecification, see line 16.
3. Reference-query instanceWithSlots, see lines 18/19
Example

```equarectangular
import ecore "http://www.nomagic.com/magicdraw/UML/2.5.1"
import ecore "http://maplembse.maplesoft.com/common/1.0" as mse

data-source Root[Model]
data-source pkg = Root/packagedElement[Package[name="Slots"]]
data-source blocks = pkg/packagedElement[Class[mse:metaclassName="SysML::Blocks::Block"]]
data-source instances = pkg/packagedElement[InstanceSpecification]

syncTable-schema BlockSchema {
  record dim [Class|mse::metaclassName="SysML::Blocks::Block"] {
    key column /name as bName
  }
}

syncTable-schema Schema (bsc: BlockSchema){
  record dim [InstanceSpecification] {
    key column /name as iName
    reference-query .mse::instanceWithSlots @ bRef
    reference-decomposition bRef = bsc {
      foreign-key column bName as cName
    }
  }
}

worksheet-template Templates(sc: Schema) {
  vertical table tab1 at (2, 1) - sc {
    key field iName
    field cName
  }
}

syncTable blockTable = BlockSchema<blocks>
syncTable syncTable = Schema<instances>(blockTable)

workbook {
  worksheet Template(syncTable)
}
```

12.4 RecursiveInstanceWithSlots

Description

The RecursiveInstanceWithSlots virtual feature does the same thing that InstanceWithSlots does but for all possible InstanceSpecifications in the tree. If a Class A is composed by other Class B and you use recursiveInstanceWithSlots to create an InstanceSpecification of Class A, MapleMBSE will also create an InstanceSpecification for Class B with Slots.
Syntax

To use recursiveInstanceWithSlots virtual feature as a column within an InstanceSpecification dimension, the syntax is as follows:

reference-query .alias::recursiveInstanceWithSlots @reference_name

This configuration line needs to be completed with a reference-decomposition that uses a Class schema, see the example for further information. Also remember that alias is the alias you assigned to the MapleMBSE ecore. For more information on assigning aliases, see Importing the MapleMBSE Ecore (page 3).

Using the RecursiveInstanceWithSlots Virtual Feature

The following example illustrates one way to use the recursiveInstanceWithSlots virtual feature:

1. Import the MapleMBSE ecore with an alias
2. Create a schema that has a dimension accessing an InstanceSpecification, see line 16.
3. Reference-query recursiveInstanceWithSlots, see lines 18/19
Example

```plaintext
import.ecore "http://www.nomagic.com/magicdraw/UML/2.5.1"
import.ecore "http://maplembse.maplesoft.com/common/1.6" as mse

data-source Root[Model]
data-source pkg = Root/packagedElement[Package[name="Slots"]]
data-source blocks = pkg/packagedElement[Class|mse::metaclassName="SysML::Blocks::Block"]
data-source instances = pkg/packagedElement[InstanceSpecification]

syntaxtable-schema BlockSchema {
    record dim [Class|mse::metaclassName="SysML::Blocks::Block"] {
        key column /name as bName
    }
}
syntaxtable-schema Schema (bsc: BlockSchema)
    record dim [InstanceSpecification] {
        key column /name as iName
        reference-query .mse::recursiveInstanceWithSlots @ bRef
        reference-decomposition bRef = bsc {
            foreign-key column bName as cName
        }
    }
}
worksheet-template Template(sc: Schema) {
    vertical table tab1 at (2, 1) = sc {
        key field iName
        field cName
    }
}
syntaxtable blockTable = BlockSchema<blocks>
syntaxtable syncTable = Schema<instances>(blockTable)

workbook {
    worksheet Template(syncTable)
}
```

12.5 AttachedFile

Description

The attachedFile virtual feature supports MagicDraw file attachments, which are accessible through comments. It downloads all the relevant file attachments, and displays hyperlinks to temporary locations in the user interface. When the user clicks on the link, the user interface will open the file.

Syntax

The syntax to use attachedFile is as follows:

```plaintext
column /mse::attachedFile as fileName
```
Using the attachedFile Virtual Feature

The following example illustrates one way to use the recursiveInstanceWithSlots virtual feature:

1. Import the MapleMBSE ecore with an alias
2. Create a schema that has a dimension accessing an ownedComment line 5, in the column level call the attached VF see line 6 in the example code

Example

syncTable-schema AttachedFileSchema {
record dim[Package] {
key column /name as PkgName
}
dim /ownedComment[Comment|mse::stereotypeNames="UML Standard Profile::MagicDraw Profile::AttachedFile"]{
column /mse::attachedFile as fileName
}
}

12.6 Slots

Description

Use the slots virtual feature to add a new slot and display all the slots recursively under a top-level instance. The slots virtual feature can also be used to delete a child slot under a top level instance and then recreate the child slot.

Syntax

The syntax to use slots is as follows:
dim /mse::slots[Slot]

Using the slots Virtual Feature

The following example illustrates one way to use the Slots virtual feature:

1. Import the MapleMBSE ecore with an alias
2. Create a schema that has a dimension accessing slots using the slots virtual feature (shown in line 6 in the slots VF get all the slots Recursively under a instance specification
Example

```plaintext
synctable-schema InstanceTable(blk : BlocksTable){
record dim [InstanceSpecification] {
key column /name as instanceName
}

dim /mse::slots[Slot] {
key reference-query .definingFeature[Property] @ dfRef
reference-decomposition dfRef = blk {
foreign-key column bName as bName
foreign-key column valName as valName
}
key column /mse::slotValue as cValue
}
```
13 Recursivity

13.1 getRecursively

Description

The getRecursively virtual feature works as a chained data source, traversing all subelements recursively under the owner data source or QPE and then filters out elements matching the qualifier and filter.

Note: This virtual feature only works for packaged elements.

Syntax

The general syntax for using the getRecursively virtual feature is as follows:

```
data-source packages = Root/packagedElement[Package|name="C3"]/getRecursively[Package]
```

Where C3 is the name of the package. All packages under C3 will be retrieved.

```
data-source packages = Root/packagedElement[Package]/getRecursively[Package]
```

In the above data source syntax example, all the packages under root are retrieved. After that, all the elements (packages) under those packages are retrieved, recursively. When you are adding a new element, in this case, it will go under one of the packages which was retrieved from the root.

Note: This feature only works with elements supporting the packageElement feature.

```
data-source packagesC3Class = Root/mse::getRecursively[Package|name="C3"]/packagedElement[Class]
```

In the syntax example above, first, getRecursively finds the packages under the model. The model may have more than one C3 package. After you have all the packages then you display classes under those packages only, not recursively.

Note that this is maybe very inefficient when the model is big, and it would be much faster to explicitly specify the path for each existing C3 package.

```
data-source packagesC3Class = Root/mse::getRecursively[Package|name="C3"]/ mse::getRecursively[Class]
```

This data source gets all packages recursively and then sorts them and shows the ones named...
C3. After that, you get all classes under these C3 packages and any of their subpackages.

**Using the getRecursively Virtual Feature**

The following example shows you how to use the ownedComments feature.

1. Import the MapleMBSE.ecore, as usual the alias used is `mse`
2. Inside a syntable-schema navigate to a `Package`
3. In the next dimension, use `/mse::getRecursively[Class]` to get all the class under the top package(Previous dim/root dim) and sub packages
4. Complete the rest of the configuration as usual: `worksheet-templates`, `synctable` and `workbook`

**Example**

In this example, the code snippet retrieves all the packages and sub packages under the package C3

```
synctable-schema PackageTable []
   record dim [Package] {
     key column /name as pName
   }
   record dim / mse:: getRecursively[Class] {
     key column /name as cName
   }
```

This feature can be used anywhere in a QPE or data source, but not at the start of QPE or data source.
```
synctable-schema PackageTable {
  record dim [Package] {
    key column /name as Name
  }
  record dim / mse:: getRecursively[Package] {
    key column /name as Name
  }
}
```
14 Constraints

SysML makes it hard to accessing the minimum and maximum constraint data in SysML can be difficult because the model is forced to use LiteralString and other elements (e.g. TimeExpression, Constraints). The main purpose of the virtual feature in this chapter is to allow the MSE file to access this data with ease and aggregate it into simple double period notation (..). Another benefit to the use of a virtual feature for working with constraint data is that the end user has fewer inputs to provide, reducing human error.

14.1 durationConstraint

Description

To display and set duration constraints, MapleMBSE provides a virtual feature that allows the simultaneous creation and editing of the min and max limits of the constraint using a simple double dot notation (for example, minConstraint..maxConstraint). In the case of both minConstraint and maxConstraint representing numerical values (in decimal or scientific notation), MapleMBSE performs a check to determine if the minConstraint value is less than or equal to the maxConstraint value.

Syntax

The general syntax for using the durationConstraint virtual feature is as follows:

```
column /mse::durationConstraint as dcValue
```

For user input, the durationConstraint virtual feature accepts numerical values as well as arbitrary string values. These values are joined by double periods (..). In addition, you can use an escape character (\) to include periods as part of the minimum and maximum constraint values.

The table below gives specific examples of both valid and invalid syntax for use of double periods.

<table>
<thead>
<tr>
<th>Constraint Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minor</td>
<td>0..18</td>
</tr>
<tr>
<td>Voter</td>
<td>18..99</td>
</tr>
<tr>
<td>TimeConfTx</td>
<td>10..60</td>
</tr>
<tr>
<td>Temp</td>
<td>cold..hot</td>
</tr>
<tr>
<td>User Input Example</td>
<td>Minimum Constraint Value</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>2..3.5</td>
<td>2</td>
</tr>
<tr>
<td>-1..3.2</td>
<td>-1</td>
</tr>
<tr>
<td>5.5..2</td>
<td>5.5</td>
</tr>
<tr>
<td>2.3 .. 2.3</td>
<td>2.3</td>
</tr>
<tr>
<td>1..3</td>
<td>1</td>
</tr>
<tr>
<td>abc..foo bar</td>
<td>abc</td>
</tr>
<tr>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>..</td>
<td></td>
</tr>
<tr>
<td>min..min</td>
<td></td>
</tr>
<tr>
<td>..max</td>
<td></td>
</tr>
<tr>
<td>escape..dots</td>
<td>escape.</td>
</tr>
<tr>
<td>invalid..input</td>
<td></td>
</tr>
<tr>
<td>2 . 5 .. 2.4</td>
<td>2.5</td>
</tr>
</tbody>
</table>

### Using the durationConstraint Virtual Feature

The following example shows you how to use the durationConstraint feature.

1. Import the MapleMBSE.ecore, as usual the alias used is mse.
2. Inside a synctable-schema navigate to a DurationConstraint, in this case /ownedRule[DurationConstraint] within an Activity.
3. Within that dimension, define a regular column using /mse::durationConstraint.
4. Complete the rest of the configuration as usual: worksheet-templates, synctable and workbook.

### Example

```plaintext
synctable-schema Schema { record dim [Activity] { ..
```
key column /name as aName
}
record dim /ownedRule[DurationConstraint] {
  key column /name as dcName
  column /mse::durationConstraint as dcValue
}
}
15 Generalization

This section contains all other virtual features that do not create elements but offer a better alternative to access and map model information.

15.1 specificClass

Description

The specificClass virtual feature provides a simple, more direct way of creating generalizations between a more generalized element and a more specialized element. The specificClass virtual feature also sets the values for the specific and general elements and then stores the generalization relationship information in the specific class.

Syntax

The general syntax for using the specificClass virtual feature is as follows:

.mse::specificClass

Using the specificClass Virtual Feature

The following example illustrates one way to use the specificClass virtual feature:

1. Import the MapleMBSE.ecore with an alias
2. Create a datasource which query the Blocks from the model 
   "../PackageElement-Class|mse::metaclassName="SysML::Blocks::Block"]"
3. In the synctable schema start from the generalized block
4. For the next dimension, use the virtual feature which uses the reference decomposition to create the generalization between the generalized block and the specific block

Example

```
synctable-schema BlockpropertiesTable(blocks: BlocksTable){
  record dim [Class|mse::metaclassName="SysML::Blocks::Block"] {
    key column /name as generalClassName
  }
  record dim .mse::specificClass[Class] @genname{
    reference-decomposition genname = blocks{
      foreign-key column BlockName as specificClassName
    }
  }
} 
```
Index

A
activity diagrams, 41
  ActivityControlFlow, 41
description, 41
syntax, 41
using, 41
ActivityObjectFlow, 43
description, 43
syntax, 43
using, 43
associations, 11
  associatedProperty, 13
description, 11
syntax, 11
using, 12
directedAssociatedProperty, 15
description, 14
syntax, 14
using, 14
nestedDirectedComposition, 19
description, 19
syntax, 19
using, 19
otherAssociatedEnd, 17
description, 16
syntax, 16
using, 16

B
blocks, 21
    getAllProperties, 25
description, 23
example, 26
syntax, 25
using, 25
propertyDefaultValue, 22
recursivePartProperties, 21
description, 21
syntax, 21

C
comments, 47
  ownedComments, 49
description, 47
syntax, 47
using, 47
connectors, 27
  connectedPropertyOrPort, 27
description, 27
syntax, 27
using, 28
otherConnectorEnd, 29
description, 29
syntax, 29
using, 29
constraints, 67
  durationConstraint, 69
description, 67
syntax, 67
using, 68

D
dependencies, 31
  clientDependencies, 31
description, 31
syntax, 31
using, 31
supplierDependencies, 33
description, 32
syntax, 33
using, 33

E
enumeration, 35
  enumerationLabel, 37
description, 36
syntax, 36
enumerationName, 35
description, 35
syntax, 35
using, 35

G
Generalization, 71
  specificClass, 71
  description, 71
  example, 71
  syntax, 71
  using, 71

R
recursivity, 63
  getRecursively, 65
  description, 63
  syntax, 63
  using, 64

S
slots, 51
  attachedFile, 59
    description, 59
    syntax, 59
    using, 60
  instanceTree, 55
    description, 53
    syntax, 54
    using, 54
  instanceWithSlots, 56
    description, 56
    syntax, 56
    using, 56
  RecursiveInstanceWithSlots, 57
    description, 57
    syntax, 58
    using, 58
slots, 61
  description, 60
  syntax, 60
  using, 60
slotValue, 51
  description, 51
  syntax, 51

using, 52
specificClass, 71
statemachines, 45
  vertexTransition, 45
    description, 45
    syntax, 45
    using, 45
stereotypes, 5
  featureName, 7
    description, 6, 7
    syntax, 7
metaclassName, 5
  description, 5
  syntax, 5
  using, 5
stereotypeNames, 9
  description, 9
  syntax, 9
  using, 9

U
util, 39
  multiplicityProperty, 39
    description, 39
    syntax, 39
    using, 39