Konfiguration von SERGe

1. Einleitung

Dieses Dokument beschreibt kurz die Konfiguration von SERGe.

Im Einleitungskapitel 1 werden einige Begriffe und wichtige Funktionen kurz erklärt. In Kapitel 2 wird auf jedes XML-Element der Konfiguration beschrieben. Kapitel 3 zeigt ein einfaches Beispiel.

1.1. Begriffe

- Fokuselement: Element für das eine Regel erstellt wird. Genauer: Das Element, auf das das Hauptaugenmerk eines Benutzer gerichtet ist, wenn er eine Änderung durch die Ausführung einer Regel bewirken möchte (z.B. bei CREATE_Class_In_Package ist Class ein Element im Fokus der Regel bzw. für den Generierungsprozess. Bei SET_Class_name ebenfalls.)
- Kontextelemente: Kontextelemente representieren „Einhängepunkte“ für eine Änderung. Bei CREATE/DELETE-Regel ist das Kontextelement das Container-Element für das zu erstellende bzw. löschende Fokuselement.
- Dangling Elements: Dies sind Elemente, welche für ein Fokuselement obligatorisch (mandatory) oder optional im Hinblick auf die einzuhaltenden Multiplizitätsrandbedingungen sind. Solche Elemente sind entweder „Kinder“ (Containments) und oder „Nachbarn“ (Non-Containments) des Fokuselements. Danglings sind in Editierregeln niemals <<create>> Knoten, sondern <<preserve>>.

1.2. Vererbungen im Metamodell und Regelanwendbarkeit

Grundsätzlich gilt für Henshin-Regeln, dass sie nicht nur für die in den Regeln verwendeten Elementtypen anwendbar sind, sondern auch für die Subtypen der definierten Elementtypen. Editierregeln kann man also auch im weitesten Sinne als „vererbar“ ansehen.

Allerdings sind Regeln für den Subtyp theoretisch überflüssig, sofern die darin enthaltenden Editierschritte sich auf EAttributes oder EReferences eines Obertyps beziehen und diese Editierschritte ebenfalls für den Obertyp in Regeln generiert werden. Diese überflüssigen Regeln können allerdings je nach Einsatzbereich der Editierregeln trotzdem von Interesse sein. Die Create-Regeln für Obertypen sind natürlich niemals überflüssig, da sie nicht für ihre Subtypen gelten. Aus diesem Grund geschieht die Generierung von Create-Regeln eines konkreten Subtyps unabhängig von den Konfigurationen zu seinem Obertypen. Für welchen Elementtyp, sofern es ein Fokusselement laut Metamodlel sein kann, Regeln generiert werden sollen, ist in <ModuleSetOptions /> zu spezifizieren (siehe 2.4.4). Genauso kann für Elementtypen festgelegt werden, ob sie selbst als Danglings in Regeln zu anderen fokussierten Elementen vorkommen dürfen und in welchem Fall (siehe 2.4.3). Da default-Werte festgelegt werden, müssen nicht alle Elemementtypen zwangsläufig separat aufgelistet werden. Zudem müssen nicht zwangsläufig alle Subtypen aufgelistet werden: Konfigurationseinstellungen zu Obertypen werden ihren nicht explizit auf der Konfiguration angegebenen Untertypen übertragen. Besitzt ein Typ eine eigene spezielle Einstellung in der Konfiguration, wird diese bevorzugt.

1.3. Reduzierung der Regelmenge

Die Regelmenge kann bei Metamodellen mit vielen und tiefen Vererbungshierarchien sehr umfangreich sein. Die Regelmenge wächst mit jeder notwendigen Ersetzung von (abstrakten) Obertypen an Dangling-Positionen in initial generierten Editierregeln: Das Postprocessing muss dabei für jede Kombination von Elementtypen an den verschiedenen Dangling-Positionen in Editierregeln eine weitere Regel erstellen; also Varianten der ursprünglichen Regel. Da eine Editierregel beliebig viele Danglings enthalten kann, kann das Kreuzprodukt beliebig umfassend und somit die zusätzlichen Regeln vielzählig sein.

Da man allerdings nicht in jedem Einsatzszenario alle Varianten benötigt, sondern nur auf die allgemeineren Regeln zurückgreifen möchte, lässt sich die Regelmenge per Konfiguration reduzieren. Dies ist innerhalb des Bereichs <SuperTypeReplacement /> und <AbstractReplacements /> für jeden Regeltyp zu konfigurieren. In beiden Bereichen gibt es folgende Optionen:

(1) reduceDanglings= true | false

(2) reduceContext= true | false

(1) bewirkt, dass die Regelmenge reduziert wird, da die Menge der Elementtypen, welche als Danglings innerhalb von Editierregeln in Frage kommen, auf die kleinstmögliche Menge reduziert wird; genauer: Die Subtypen fallen aus der Menge, sofern die Syntax des Metamodells dadurch nicht verletzt ist. Da Obertypen in den Regeln an den Dangling-Positionen verwendet werden, sind die Regeln auch anwendbar auf Instanzen, in denen Subtypen an der gefragten Stelle auftreten. Durch die Reduzierung der Elemente müssen weniger Varianten von Regeln generiert werden, welche sich aus den Kreuzprodukten der möglichen Elementtypen in den Dangling-Positionen ergeben (in einer Editierregeln können [0..*] Dangling-Elements auftreten).

(2) bewirkt analog die Reduzierung der Regelmenge durch Rreduzierung der Menge von Elementtypen, welche als Kontextelement innerhalb von Editierregeln auftreten können. Auch hier werden, sofern syntaktisch durch das Metamodell erlaubt, Obertypen verwendet. Dies spart ebenfall eine Generierung von Varianten, die durch Ersetzung des im Metamodell als Kontext spezifizierten Obertyp durch seine verschiedenen Subtypen.

2. Konfigurationselemente

2.1. MetaModel

Die verwendeten Metamodelle festlegen und ihre Verwendung konfigurieren.

2.1.2. MainModel

Legt das Metamodell fest, für das Regeln erstellt werden sollen.
Alle Modelle, von denen das Modell abhängig ist, müssen als *RequiredModel* angegeben werden.
Attribute:

- nsUri (CDATA): Namespace-URI des Metamodells
- isProfile (true|false): true, wenn es sich um ein UML-Profil handelt. Ist diese Option falsch gesetzt werden ungültige Regeln erzeugt.

2.1.2. RequiredModel*

Fügt ein Modell hinzu, von welchem das Hauptmodell abhängig ist.
Attribute:

- nsUri: (CDATA): Namespace-URI des Modells

2.1.3 MaskedClassifiers (optional)

In manchen Metamodellen kommt es vor, dass Typen nicht EClasses defniert sind, sondern dass andere EClasses ein EAttribut mit EnumerationLiteal-Typ beinhaltet, welches eine verkappte Typinformation darstellt. Dies ist beispielsweise in der UML Version 2.4 der Fall: Es gibt keinen ForkPseudoState oder JoinPseudoState. Dort gibt es nur eine EClass PseudoState mit einem type-Attribut, welches auf ein PseudoStateKind EnumerationLiteral zeigt.
Diese verhüllte Typinformation lässt sich nicht automatisch im Generierungsalgorithmus als solche erkennen. Daher wird dem Nutzer durch die Konfiguration von sogenannten „Masken“ die Möglichkeit geboten, solche Verhüllungen bzw. „Masken“ zu identifizieren und für den Generierungsprozess zu defnieren. Die „Masken“ werden m.a.W. wie eigene Typen behandelt werden. Dadurch lassen sich speziell für einen makierten Typ Regeln generieren. Im Obigen Beispiel bedeutet dies das zusätzliche Generieren von MOVE_ForkPseudoState… anstatt nur MOVE_Pseudostate… MaskedClassifiers beinhaltet eine Liste von *Masks*.

2.1.3.1. Mask*

Definiert eine Maske.

Attribute:

- name (CDATA): Name er Maske, die sich durch die anderen Attribute definiert wird
- eClassifier (CDATA): Typ aus einem angegeben Metamodell.
- eAttriibute (CDATA): Name eines Attributs von eClassifier
- eAttriibuteValue (CDATA): Gültiger Attributwert von eEAttribute (auch Literale möglich)

2.1.4. RootModelElement (optional)

?
Attribute:

- name (CDATA): Name eines Elements des Metamodells oder leer

- canBeNested (true|false): ?

2.2. Transformations

Bestimmt, welche Typen von Regeln erstellt werden. Einige Regeln haben zusätzliche Einstellungsmöglichkeiten. Wird ein Regeltyp nicht angegeben, wird „enable“ als „false“ angenommen. Liste von *Transformation*.

2.2.1. (Transformation)*

Element-Auswahl:
CREATE, DELETE, ATTACH, DETACH, MOVE, MOVE_UP, MOVE_DOWN, SET_ATTRIBUTE, UNSET_ATTRIBUTE, SET_REFERENCE, UNSET_REFERENCE, ADD, REMOVE, CHANGE_LITERAL, CHANGE_REFERENCE
Attribute:
- enable (true|false): Wenn auf „true“ gesetzt werden Regeln dieses Typs generiert

Zusätzliche Attribute ATTACH und DETACH:
- useSimpleNames (true|false): Wenn „true“ werden die Namen der erweiterten Elemente nicht in den Namen der ATTACH- bzw. DETACH-Regel aufgenommen, solange dadurch die Namen eindeutig bleiben.

Zusätzliche Attribute MOVE:
- allowReferenceSwitching (true|false): Falls true, können Move Regeln erstellt werden, bei denen das Zielelternelement einen anderen Typ hat als das vorige Elternelement. Dabei darf sich dann auch die Containment-Referenz gemäß Metamodellsyntax ändern.
- allowReferenceCombinations (true|false): Wenn „true“ werden Regeln vom Typ MOVE_REFERENCE_COMBINATION erzeugt. Diese Regeln werden durch Berücksichtigung der Kreuzprodukte aus allen möglichen Elternelementtypen (einschließlich Subtypen) und Containment-Referenztypen gebildet, zwischen denen Elemente verschoben werden können.

Zusätzliche Attribute CHANGE_LITERAL:
- allowLiteralSwitching (true|false):Falls „true“ werden Regeln vom Typ CHANGE_LITERAL generiert. Diese Regel representieren das Ändern eines Attributwerts ausgehend von jeglichem Literalwert zu jedem anderen möglichen Literalwert.

2.3. ConsistencyLevel

Einstellungen zur Konsistenz der erstellten Regeln.

2.3.1. CreateMandatoryChildren

Bestimmt, ob benötigte (mandatory) Kinder in den Regeln generiert werden sollen.

Attribute:
- enable (true|false): Wenn „true“ werden mandatory children erstellt

2.3.2. CreateMandatoryNeighbours

Bestimmt, ob benötigte (mandatory) Nachbarn in den Regeln generiert werden sollen.

Attribute:
- enable (true|false): Wenn „true“ werden mandatory neighbours erstellt

2.3.3. CreateMultiplicityPreconditions

Bestimmt, ob in den Regeln <<forbid>>/<<require>> Knoten und Kanten generiert werden sollen, welche ein Muster von Vorbedingungen für eine Änderung defnieren. Vorbedingungen sind Einschränkungen, die durch Multiplizitäten an eingehenden oder ausgehenden Referenzen im Metamodell definiert sind.

Attribute:

- enable (true|false): Wenn „true“ werden Knoten erstellt, die Einschränkungen bezüglich von Multiplizitäten umsetzen

2.4. CompletnessLevel

Definiert die Vollständigkeit einzelner Regeln und der Regelmenge.

2.4.1. AbstractReplacements

Da von abstrakten Elementen keine Instanzen erstellt werden dürfen, dürfen auch die generierten Regeln ein solchen Verhalten nicht darstellen. Abstrakte Typen dürfen aber durchaus als Dangling Elements oder als Kontextelemente innerhalb von Regeln auftreten.

Um alle möglichen Verhaltensweisen mit Regeln in den Editierregelsets abzubilden, müssen auch Regeln enthalten sein bzw. generiert werden, in denen mandatory, dangling Elements abstrakte Typen haben. Dies ist aber nicht zwangsläufig immer gewünscht, da man u.U. genauer steuern möchte, welche konkreteren Ausprägungen in den Danglings vorkommen sollen. Aus diesem Grund besitzt SERGe einen Postprocessing-Algorithmus zur Ersetzung von abstrakten, mandatory, dangling Elements in bereits generierten Regeln. Diese Erstetzung lässt sich für jeden Regeltyp an oder abschalten. Bei der Ersetzung werden, je nach Fall, Regelinhalte ersetzt oder sogar zusätzliche Varianten von Regeln generiert (da die Ersetzung u.U. auch rekursiv neue mandatory Danglings oder Preconditions fordert). Der Algorithmus ist standardmäßig angeschaltet, lässt sich aber wie im folgenden beschrieben ausschalten, so dass die Regelmenge bzw. die Regelinhalte hinsichtlich der (rekursiv hinzugekommenen) danglings reduziert wird: reduceDanglings=true schaltet die Ersetzung aus.

Wenn ein Regeltyp nicht angegeben ist, wird das Attribut „reduceDanglings“ für ihn als „false“ angenommen. Liste von *AbstractReplacement*.

2.4.1.1. AbstractReplacement*

Attribute:

- ruleType (CREATE_DELETE | MOVE | MOVE_UP | MOVE_DOWN | SET_UNSET_ATTRIBUTE | SET_UNSET_REFERENCE | ADD_REMOVE | CHANGE_REFERENCE | CHANGE_LITERAL | ATTACH_DETACH): Regeln auf die sich diese Einstellung bezieht
- reduceDanglings (true|false): Wenn „true“, dann werden für Danglings **keine** Regelvarianten generiert, sofern der ursprüngliche Elementtyp abstrakt ist.

2.4.2. SupertypeReplacements

Analog zu AbstractReplacements kann auch eine SupertypReplacement gefordert oder ausgeschaltet werden. SupertypReplacements bezieht sich nur auf Obertypen die bereits konkret

sind. Hier werden ebenfalls Typen in den Regelinhalten ersetzt (sofern der Obertyp nicht gewünscht war lt. Config) oder/und neue Varianten von Regeln generiert. Außerdem lässt sich hier ebenfalls einstellen, ob die Typersetzung bzw. Variantengenerierung auch die Kontextelemente einer Regel betrachten bzw. ersetzen soll.

Wird ein Regeltyp nicht angegeben, werden die Attribute „reduceDanglings“ und „reduceContext“ als „false“ angenommen. Liste von *SupertypeReplacement*.

2.4.2.1. SupertypeReplacement*

Attribute:

- ruleType (CREATE_DELETE | MOVE | MOVE_UP | MOVE_DOWN | SET_UNSET_ATTRIBUTE | SET_UNSET_REFERENCE | ADD_REMOVE | CHANGE_REFERENCE | CHANGE_LITERAL | ATTACH_DETACH): Regeln auf die sich diese Einstellung bezieht.
- reduceDanglings (true|false): Wenn „true“, dann werden für Danglings *keine* Varianten für Untertypen des Typs erstellt.
- reduceContext (true|false): Wenn „true“, dannwerden *keine* Varianten für Untertypen des Kontextelements erstellt.

2.4.3. ModuleInteralOptions

Einstellungen, die einzelne Regeln betreffen.

2.4.3.1. CreateOptionalAttributes

Bestimmt nur benötigte (mandatory) oder auch optionale Attribute in den Regeln erstellt werden sollen.

Attribute:

- enable (true|false): Wenn „true“ werden auch alle optionale Attribute in die Regeln erstellt. (also jene, deren lowerbound-Multiplizität im Metamodell 0 ist).

2.4.3.2. DanglingNodeSettings

Einstellungen für Danglings.

Attribute:

- defaultTypeInclusion (never|ifRequired|always): Wert für „include“, der für nicht aufgelistete Typen impliziert wird. Siehe *DanglingNodeSetting.*

2.5.3.2.1. DanglingNodeSetting*

Legt explizit fest, wann ein Typ als Dangling auftreten darf.
Attribute:

- type (CDATA): Name eines Typs in einem angegeben Metamodell
- include (never|ifRequired|always):
  - never: Der Typ soll niemals als Dangling auftreten
  - always: Der Typ darf als Dangling auftreten
  - ifRequired: Der Typ soll nur als Dangling auftreten, wenn er (je nach Regeltyp) von einem direkten/indirekten Kind oder Subtyp eines als „always“ deklarierten Typ benötigt wird (z.B. wenn er ein mandatory Neighbor für andere lt. Metamodell darstellt; m.a.W. wenn eine Referenz auf diesen Typen mit einer LowerBound-Multiplizität von 1 oder größer zeigt.). Bei Ermittlung von benötigten Typen wird rekursiv vorgegangen. Dabei

werden auch Oberklassen und (bei abstrakten Klassen) Unterklassen berücksichtigt. Falls ein Typ mehrere Obertypen hat, haben die Optionen „ALWAYS“ und „NEVER“ Vorrang vor „IF_REQUIRED“. Sollte ein Konflikt zwischen „ALWAYS“ und „NEVER“ auftreten wird die vom Nutzer definierte Defaulteinstellung verwendet.

2.4.4. ModuleSetOptions

Einstellungen, die die Menge der generierten Regeln betreffen.

2.4.4.1. ModuleGenerationSettings

Legt fest, für welche Typen Regeln erstellt werden, d.h. diese als Fokuselement auftreten können. In der Regel betrifft dies nur Typen des Haupt-Metamodells.

Attribute:

- defaultGenerationCase (never|ifRequired|always): Wert für „case“, der für nicht aufgelistete Typen impliziert wird. Siehe *GenerateModulesFor.*

2.5.4.1.1. GenerateModulesFor*

Legt fest ob ein Typ als Fokuselement verwendet werden soll, d.h. Regeln für ihn generiert werden sollen.

Attribute:

- type (CDATA): Name eines Typs im Metamodell
- case (never|ifRequired|always):
  - never: Für den Typ werden keine Regeln generiert.
  - always: Für den Typ werden Regeln erstellt.
  - ifRequired: Für den Typ werden nur Regeln generiert, wenn er von einem anderen Typ oder von einem UML-Stereotyp direkt oder indirekt benötigt wird (m.a.W. wenn eine Referenz auf diesen Typen mit einer LowerBound-Multiplizität von 1 oder größer zeigt.). Bei Ermittlung von benötigten Typen wird rekursiv vorgegangen. Dabei werden auch Oberklassen und (bei abstrakten Klassen) Unterklassen berücksichtigt. Ein Auslassen des Typ bzw. der generierten Regeln kann bei der Regelausführung die Konsistenz des Modells verletzen oder viel eher sogar andere Regeln gar nicht erst anwendbar machen, da im Modell die notwendige Elemente fehlen. Falls ein Typ mehrere Obertypen hat, haben die Optionen „ALWAYS“ und „NEVER“ Vorrang vor „IF_REQUIRED“. Sollte ein Konflikt zwischen „ALWAYS“ und „NEVER“ auftreten wird die vom Nutzer definierte Defaulteinstellung verwendet.

3. Beispiel

Dieses Beispiel zeigt eine mögliche Konfiguration für UML-Statemachines.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Config SYSTEM "org.sidiff.editrule.generator.serge.config.dtd" >

<Config>
 <MetaModel>
    <MainModel nsUri="http://www.eclipse.org/uml2/4.0.0/UML" isProfile="false" />
    <RequiredModel nsUri="http://www.eclipse.org/uml2/4.0.0/Types" />
    <RequiredModel nsUri="http://www.eclipse.org/emf/2002/Ecore" />
    <MaskedClassifiers>
      <Mask name="InitialPseudostate" eClassifier="Pseudostate" eAttribute="kind"
          eAttributeValue="initial" />
      <Mask name="ChoicePseudostate" eClassifier="Pseudostate" eAttribute="kind"
          eAttributeValue="choice" />
      <Mask name="DeepHistoryPseudostate" eClassifier="Pseudostate" eAttribute="kind"
          eAttributeValue="deepHistory" />
      <Mask name="ShallowHistoryPseudostate" eClassifier="Pseudostate" eAttribute="kind"
          eAttributeValue="shallowHistory" />
      <Mask name="JoinPseudostate" eClassifier="Pseudostate" eAttribute="kind"
          eAttributeValue="join" />
      <Mask name="ForkPseudostate" eClassifier="Pseudostate" eAttribute="kind"
          eAttributeValue="fork" />
      <Mask name="JunctionPseudostate" eClassifier="Pseudostate" eAttribute="kind"
          eAttributeValue="junction" />
      <Mask name="EntryPointPseudostate" eClassifier="Pseudostate" eAttribute="kind"
          eAttributeValue="entryPoint" />
      <Mask name="ExitPointPseudostate" eClassifier="Pseudostate" eAttribute="kind"
          eAttributeValue="exitPoint" />
      <Mask name="TerminatePseudostate" eClassifier="Pseudostate" eAttribute="kind"
          eAttributeValue="terminate" />
    </MaskedClassifiers>
    <RootModelElement name="StateMachine" canBeNested="true" />
</MetaModel>

<Transformations>
          <CREATE enable="true" />
          <DELETE enable="true" />
          <MOVE enable="true" allowReferenceSwitching="true" allowReferenceCombinations="true" />
          <MOVE_UP enable="true" />
          <MOVE_DOWN enable="true" />
          <SET_ATTRIBUTE enable="true" />
          <UNSET_ATTRIBUTE enable="true" />
          <SET_REFERENCE enable="true" />
          <UNSET_REFERENCE enable="true" />
          <ADD enable="true" />
          <REMOVE enable="true" />
          <CHANGE_LITERAL enable="true" allowLiteralSwitching="true" />
          <CHANGE_REFERENCE enable="true" />
</Transformations>

<ConsistencyLevel>
          <CreateMandatoryChildren enable="true" />
          <CreateMandatoryNeighbours enable="true" />
          <CreateMultiplicityPreconditions enable="true" />
</ConsistencyLevel>

<CompletnessLevel>
  <AbstractReplacements>
          <AbstractReplacement ruleType="CREATE_DELETE" reduceDanglings="true" />
          <AbstractReplacement ruleType="ADD_REMOVE" reduceDanglings="true" />
          <AbstractReplacement ruleType="MOVE" reduceDanglings="true" />
          <AbstractReplacement ruleType="MOVE_UP" reduceDanglings="true" />
          <AbstractReplacement ruleType="MOVE_DOWN" reduceDanglings="true" />
          <AbstractReplacement ruleType="SET_UNSET_ATTRIBUTE" reduceDanglings="true" />
          <AbstractReplacement ruleType="SET_UNSET_REFERENCE" reduceDanglings="true" />
          <AbstractReplacement ruleType="CHANGE_LITERAL" reduceDanglings="true" />
          <AbstractReplacement ruleType="CHANGE_REFERENCE" reduceDanglings="true" />
   </AbstractReplacements>
```

```
 <SupertypeReplacements>
    <SupertypeReplacement ruleType="CREATE_DELETE" reduceDanglings="true"
                          reduceContext="true" />
    <SupertypeReplacement ruleType="ADD_REMOVE" reduceDanglings="true" reduceContext="true" />
    <SupertypeReplacement ruleType="MOVE" reduceDanglings="true" reduceContext="true" />
    <SupertypeReplacement ruleType="MOVE_UP" reduceDanglings="true" reduceContext="true" />
    <SupertypeReplacement ruleType="MOVE_DOWN" reduceDanglings="true" reduceContext="true" />
    <SupertypeReplacement ruleType="SET_UNSET_ATTRIBUTE" reduceDanglings="true"
                         reduceContext="true" />
    <SupertypeReplacement ruleType="SET_UNSET_REFERENCE" reduceDanglings="true"
                          reduceContext="true" />
    <SupertypeReplacement ruleType="CHANGE_LITERAL" reduceDanglings="true"
                         reduceContext="true" />
    <SupertypeReplacement ruleType="CHANGE_REFERENCE"  reduceDanglings="true"
                         reduceContext="true" />
 </SupertypeReplacements>

<ModuleInternalOptions>
    <CreateOptionalAttributes enable="true" />
    <DanglingNodeSettings defaultTypeInclusion="ifRequired">
      <DanglingNodeSetting type="Activity" include="always" />
      <DanglingNodeSetting type="Behavior" include="always" />
      <DanglingNodeSetting type="Enumeration" include="always" />
      <DanglingNodeSetting type="EnumerationLiteral" include="always" />
      <DanglingNodeSetting type="FinalState" include="always" />
      <DanglingNodeSetting type="LiteralBoolean" include="always" />
      <DanglingNodeSetting type="LiteralString" include="always" />
      <DanglingNodeSetting type="OpaqueBehavior" include="always" />
      <DanglingNodeSetting type="Package" include="always" />
      <DanglingNodeSetting type="Pseudostate" include="always" />
      <DanglingNodeSetting type="Region" include="always" />
      <DanglingNodeSetting type="State" include="always" />
      <DanglingNodeSetting type="StateMachine" include="always" />
      <DanglingNodeSetting type="Transition" include="always" />
      <DanglingNodeSetting type="Constraint" include="always" />
   </DanglingNodeSettings>
</ModuleInternalOptions>

<ModuleSetOptions>
    <ModuleGenerationSettings   defaultGenerationCase="ifRequired">
      <GenerateModulesFor type="Activity" case="always" />
      <GenerateModulesFor type="Behavior" case="always" />
      <GenerateModulesFor type="Enumeration" case="always" />
      <GenerateModulesFor type="EnumerationLiteral" case="always" />
      <GenerateModulesFor type="FinalState" case="always" />
      <GenerateModulesFor type="LiteralBoolean" case="always" />
      <GenerateModulesFor type="LiteralString" case="always" />
      <GenerateModulesFor type="OpaqueBehavior" case="always" />
      <GenerateModulesFor type="Package" case="always" />
      <GenerateModulesFor type="Pseudostate" case="always" />
      <GenerateModulesFor type="Region" case="always" />
      <GenerateModulesFor type="State" case="always" />
      <GenerateModulesFor type="StateMachine" case="always" />
      <GenerateModulesFor type="Transition" case="always" />
      <GenerateModulesFor type="Constraint" case="never" />
   </ModuleGenerationSettings>
</ModuleSetOptions>
</CompletnessLevel>

</Config>
```