Translation Rules

High Level

$\quad | \quad [Expr, Param, Identifier, Field]$

$SCJBlock == \text{seq } Expr$
$Params == \text{seq } Param$
$SCJMethSig == Name \times Params$
$SCJMethod == SCJMethSig \times SCJBlock$
$Methods == \text{seq } SCJMethod$
$Fields == \text{seq } Field$
$SCJClass == Identifier \times Fields \times Methods$
$SCJProg == \text{seq } SCJClass$

$\quad | \quad [Action, CircVar, CircParam, CircName, CircType,$
$\quad \qquad CircExpression, Paragraph, Framework, ChannelDefinition,$
$\quad \qquad ChanSetDefinition, ProcDefinition]$

$CircActions == \text{seq } Action$
$CircActions == \text{seq } CircAction$
$CircState == \text{seq } CircVar$
$CircParams == \text{seq } CircParam$
$CircProcess == CircName \times CircParams \times CircState \times CircActions$
$CircParagraph ::= Paragraph \mid ChannelDefinition \mid ChanSetDefinition \mid ProcDefinition$
$CircusProg == \text{seq } CircParagraph$

$CircProcess == ProcDefinition$
$OhCircusClass == ClassDefinition$
$ChanTuple == (\text{seq } ChannelDefinition, \text{seq } ChanSetDefinition)$
$Channels == ChanTuple$
$MCBChans == ChanTuple$
$MCBActions == \text{seq } Action$
$ClassTuple == (CircProcess, OhCircusClass, Channels, MCBChans, MCBActions)$

$\quad | \quad TransSCJProg : SCJProg \nrightarrow (CircusProg, Framework)$
$\overline{\quad\qquad\qquad\qquad\qquad\qquad}$
$\quad | \quad \forall scjProg : SCJProg \mid$
$\quad | \quad \qquad TransSCJProg(scjProg) = (TransClasses(seqC), Framework)$

$ProcChannels == (\text{seq } ChannelDefinition, \text{seq } ChanSetDefinition)$
$Channels == ProcChannels$
$MCBChans == ProcChannels$
$MCBActions == \text{seq } Action$

$TransClasses : \text{seq } SCJClass \nrightarrow \text{seq } ClassTuple$

$\forall\, classes : \text{seq } SCJClass \bullet$
$\quad \forall\, classSCJClass \bullet$
$\qquad TransClasses(\langle class \rangle) = TransClass(class)\ \wedge$
$\qquad TransClasses(\langle class \rangle \frown classes) = \langle TransClass(c) \rangle \frown TransClasses(classes)$

$TransClass : SCJClass \nrightarrow ClassTuple$

$\forall\, class : SCJClass \mid \exists\, meths : \text{seq } Methods \bullet meths = MethodsOf(class) \bullet$
$\quad TransClass(class) =$
$\qquad (TransProc(class), TransOhClass(class), TransChans(meths), TransMCBChan(meths))$

$TransMeths : \text{seq } SCJMethod \nrightarrow \text{seq } Action$

$\forall\, meths : \text{seq } SCJMethod \bullet$
$\quad \forall\, meth : SCJMethod \bullet$
$\quad TransMeths(\langle meth \rangle) = TransMeth(meth)\ \wedge$
$\qquad TransMeths\langle meth \rangle \frown meths) = \langle TransMeth(meth) \frown TransMeths(meths) \rangle$

$TransMeth : SCJMethod \nrightarrow Action$

$\forall\, m : SCJMethod \bullet$
$\quad \exists\, ms : SCJMethSig;\ b : SCJBlock;\ p : Params \mid$
$\qquad TransMeth(m) = (TransMethSig(ms), TransBlock(b), TransParams(p))$

$TransParams : Params \nrightarrow CircParams$

$\forall\, params : Params \bullet$
$\quad \forall\, param : Param \bullet$
$\quad TransParams(\langle param \rangle) = TransParam(param)$
$\qquad TransParams(\langle param \rangle \frown params) = \langle TransParam(param) \frown TransParams(params) \rangle$

$TransParam : Param \nrightarrow CircParam$

$TransBlock : SCJBlock \nrightarrow CircBlock$

$\forall\, block : SCJBlock \bullet$
$\quad \forall\, e : Expr \bullet$
$\quad TransBlock(\langle e \rangle) = TransExpr(e)\ \wedge$
$\qquad TransBlock(\langle e \rangle \frown block) = \langle TransExpr(e) \frown TransBlock(block) \rangle$

Low Level

- *Method* : *MethodDeclaration* $\nrightarrow$ (*Name*, *Params*, *ReturnType*, *Body*) : translates an active application method into a **Circus** action

- *DataMethod* : *MethodDeclaration* $\nrightarrow$: translates data methods into an **OhCircus** method

- *MethodBody* : *Block* $\nrightarrow$ seq *CircExpression* : translates a Java block, for example a method body

- *Registers* : *Block* $\nrightarrow$ seq *Name* : extracts the Names of the schedulables registered in a Java block

- *Returns* : *Block* $\nrightarrow$ seq *Name* : extracts the Names of the variables retuned in a Java block

- *Variable* : (*Name*, *Type*, *InitExpression*) $\nrightarrow$ (*CircName*, *CircType*, *CircExpression*) : translates a variable

- *Parameters* : (*Name*, *Params*, *ReturnType*, *Body*) $\nrightarrow$ seq *CircParam* : translates a list of method parameters

- $[\![Name]\!]_{name}$: translates the *name* to a Z identifier

- $[\![varType]\!]_{type}$: translates types

- $[\![expr]\!]_{expression}$: translates expressions

Auxiliary Functions

- *IdOf*(*name*): yields the identifier of a component called *name*

- *ObjectIdOf*(*name*): yields the identifier of the *Object* process of a component called *name*

- *ThreadIdOf*(*name*): yields the identifier of the *Thread* process of a component called *name*

- *MethodName*(*method*): yields the method name of *method*

Pattern Matching Rules

Safelet

```
1  public class Identifier implements Safelet
2  {
3     FieldDeclaration_1
4     . . .
5     FieldDeclaration_n
6
7     ConstructorDeclaration
8
9     initializeApplication
10
11    getSequencer
12
13    AppMeth_1
14    . . .
15    AppMeth_n
16 }
```

$$\textbf{process} \; [\![Identifier]\!]_{Name} \; App \; \widehat{=} \; [\![\; [\![\; ConstructorDeclaration \;]\!]_{Method} \;]\!]_{Parameters} \; \textbf{begin}$$

State _____
  $this : \text{ref} \; [\![Identifier]\!]_{name} \; Class$
_____

state $State$

Init _____
  $State'$
  $this := \textbf{new} \; [\![Identifier]\!]_{name} \; Class()$
_____

$$InitializeApplication \; \widehat{=}$$
$$\begin{pmatrix} initializeApplicationCall \longrightarrow \\ [\![\; [\![\; InitializeApplication \;]\!]_{Method} \;]\!]_{MethBody} \\ initializeApplicationRet \longrightarrow \\ \textbf{Skip} \end{pmatrix}$$

$$GetSequencer \; \widehat{=}$$
$$\begin{pmatrix} getSequencerCall \longrightarrow \\ getSequencerRet \, ! \; [\![\; GetSequencer \;]\!]_{Returns} \longrightarrow \\ \textbf{Skip} \end{pmatrix}$$

$$[\![AppMeth\_1]\!]_{Method}$$

$$. . .$$

$$[\![AppMeth\_n]\!]_{Method}$$

$Methods \;\widehat{=}$

$$
\begin{pmatrix}
GetSequencer \\
\square \\
InitializeApplication \\
\square \\
MethName(AppMeth\_1) \\
\square \\
\ldots \\
MethName(AppMeth\_n) \\
\ldots
\end{pmatrix}
\; ; \; Methods
$$

$\bullet \; (Init \; ; \; Methods) \; \triangle(end\_safelet\_app \longrightarrow \textbf{Skip})$

end

Mission Sequencer

```
1  public class Identifier extends MissionSequencer
2  {
3      FieldDeclaration_1
4      . . .
5      FieldDeclaration_n
6
7      ConstructorDeclaration
8
9      getNextMission
10
11     AppMeth_1
12     . . .
13     AppMeth_n
14 }
```

$$\textbf{process} \, [\![Identifier\,]\!]_{Name} \, App \, \widehat{=} \, [\![\, [\![\, ConstructorDeclaration \,]\!]_{Method} \,]\!]_{Parameters} \, \textbf{begin}$$

$\underline{\quad State \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad}$
$\quad\quad this : \mathrm{ref}\,[\![Identifier\,]\!]_{name}\,Class$
$\underline{\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad}$

state $State$

$\underline{\quad Init \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad}$
$\quad\quad State'$
$\quad\quad this := \textbf{new}\,[\![Identifier\,]\!]_{name}\,Class()$
$\underline{\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad}$

$$GetNextMission \, \widehat{=} \, \textbf{var}\, ret : MissionID \, \bullet$$
$$\begin{pmatrix} getNextMissionCall\,.\,IdOf(Identifier) \longrightarrow \\ ret := this\,.\,getNextMission(); \\ getNextMissionRet\,.\,IdOf(Identifier)\,!\,ret \longrightarrow \\ \textbf{Skip} \end{pmatrix}$$

$$[\![AppMeth\_1]\!]_{Method}$$

. . .

$$[\![AppMeth\_n]\!]_{Method}$$

$$Methods \, \widehat{=}$$
$$\begin{pmatrix} GetNextMission \\ \square \\ MethName(AppMeth\_1) \\ \square \\ MethName(AppMeth\_n) \\ \ldots \end{pmatrix} ; \; Methods$$

$$\bullet \; (Init\,;\; Methods) \;\; \triangle(end\_sequencer\_app\,.\,IdOf(Identifier) \longrightarrow \textbf{Skip})$$

end

Mission

```
 1  public class Identifier extends Mission
 2  {
 3      FieldDeclaration_1
 4      . . .
 5      FieldDeclaration_n
 6
 7      ConstructorDeclaration
 8
 9      initialize
10
11      cleanUp
12
13      AppMeth_1
14      . . .
15      AppMeth_n
16  }
```

$$\mathbf{process}\; [\![\, Indentifier \,]\!] \; App \;\widehat{=}\; [\![\; [\![\; ConstructorDeclaration \;]\!]_{Method} \;]\!]_{Parameters} \;\mathbf{begin}$$

$\underline{\;State\;}$
$\quad this : \mathrm{ref}\; [\![\, Identifier \,]\!]_{name}\; Class$

$\mathbf{state}\; State$

$\underline{\;Init\;}$
$\quad State'$
$\quad this := \mathbf{new}\; [\![\, Identifier \,]\!]_{name}\; Class()$

$InitializePhase \;\widehat{=}$
$$\begin{pmatrix} initializeCall . IdOf(Indentifier) \longrightarrow \\ [\![\, initialize \,]\!]_{Registers}\; initializeRet . IdOf(Indentifier) \longrightarrow \\ \mathbf{Skip} \end{pmatrix}$$

$CleanupPhase \;\widehat{=}$
$$\begin{pmatrix} cleanupMissionCall . IdOf(Indentifier) \longrightarrow \\ cleanupMissionRet . IdOf(Indentifier)\,!\,\mathbf{True} \longrightarrow \\ \mathbf{Skip} \end{pmatrix}$$

$[\![\, AppMeth\_1 \,]\!]_{Method}$

$\quad$. . .

$[\![\, AppMeth\_n \,]\!]_{Method}$

$$Methods \;\widehat{=}\; \begin{pmatrix} InitializePhase \\ \square \\ CleanupPhase \\ \square \\ MethName(AppMeth\_1) \\ \square \\ MethName(AppMeth\_n) \\ \ldots \end{pmatrix} \; ; \; Methods$$

$\bullet \; (\mathit{Init} \; ; \; \mathit{Methods}) \; \triangle (\mathit{end\_mission\_app} \, . \, \mathit{IdOf}\,(\mathit{Identifier}) \longrightarrow \mathbf{Skip}$

end

Handlers

```
1  class Identifier extends HandlerType
2  {
3     FieldDeclaration_1
4     . . .
5     FieldDeclaration_n
6
7     ConstructorDeclaration
8
9     handleAsyncEvent
10
11    AppMeth_1
12    . . .
13    AppMeth_n
14 }
```

$$\textbf{process}\,[\![\,PName\,]\!]\,App \;\widehat{=}\; [\![\,[\![\,ConstructorDeclaration\,]\!]_{Method}\,]\!]_{Parameters}\;\textbf{begin}$$

$$\begin{array}{|l}\hline State \\ \hline \quad this : \mathrm{ref}\,[\![\,Identifier\,]\!]_{name}\;Class \\ \hline \end{array}$$

state *State*

$$\begin{array}{|l}\hline Init \\ \hline \quad State' \\ \quad this := \textbf{new}\,[\![\,Identifier\,]\!]_{name}\;Class() \\ \hline \end{array}$$

$$handleAsyncEvent \;\widehat{=}$$
$$\left(\begin{array}{l} handleAsyncEventCall\,.\,IdOf(PName) \longrightarrow \\ [\![\,[\![\,HandleAsyncBody\,]\!]_{Method}\,]\!]_{MethBody}; \\ handleAsyncEventRet\,.\,IdOf(PName) \longrightarrow \\ \textbf{Skip} \end{array}\right)$$

$$[\![\,AppMeth\_1\,]\!]_{Method}$$

. . .

$$[\![\,AppMeth\_n\,]\!]_{Method}$$

$$Methods \;\widehat{=}$$
$$\left(\begin{array}{l} handleAsyncEvent \\ \square \\ MethName(AppMeth\_1) \\ \square \\ MethName(AppMeth\_n) \\ \ldots \end{array}\right)\;;\;\; Methods$$

$$\bullet\,(Init\,;\;Methods)\;\;\triangle(end\_\,[\![\,HandlerTypeIdOf(PName)\,]\!] \longrightarrow \textbf{Skip})$$

end

Managed Thread

```
1  public class Identifier extends ManagedThread
2  {
3     FieldDeclaration_1
4     . . .
5     FieldDeclaration_n
6
7     ConstructorDeclaration
8
9     run
10
11    AppMeth_1
12    . . .
13    AppMeth_n
14 }
```

$\textbf{process}\ [\![PName\ ]\!]\ App \mathrel{\widehat{=}} [\![\ [\![\ ConstructorDeclaration\ ]\!]_{Method}\ ]\!]_{Parameters}\ \textbf{begin}$

State
$this : \mathrm{ref}\ [\![Identifier\ ]\!]_{name}\ Class$

---

state $State$

Init
$State'$
$this := \textbf{new}\ [\![Identifier\ ]\!]_{name}\ Class()$

$Run \mathrel{\widehat{=}}$
$\begin{pmatrix} runCall\,.\,IdOf(PName) \longrightarrow \\ [\![\ [\![\ run\ ]\!]_{Method}\ ]\!]_{MethBody}; \\ runRet\,.\,IfOf(PName) \longrightarrow \\ \textbf{Skip} \end{pmatrix}$

$[\![AppMeth\_1]\!]_{Method}$

$\quad \cdots$

$[\![AppMeth\_n]\!]_{Method}$

$Methods \mathrel{\widehat{=}}$
$\begin{pmatrix} Run \\ \square \\ MethName(AppMeth\_1) \\ \square \\ MethName(AppMeth\_n) \\ \cdots \end{pmatrix}\ ;\ Methods$

$\bullet\ (Init\ ;\ Methods)\ \triangle(end\_managedThread\_app\,.\,IdOf(PName) \longrightarrow \textbf{Skip})$

end

Data Class

class $[\![PName \,]\!]_{name}$ *Class* $\,\widehat{=}\,$ **begin**

---
state *State*
$[\![VarName]\!]_{name} : [\![VarType]\!]_{type}$

---

state *State*

---
initial *Init*
State$'$

$[\![VarName]\!]'_{name} = [\![VarInit]\!]_{expression}$

$[\![DataMeth1]\!]_{dataMeth}$
$[\![DataMeth2]\!]_{dataMeth}$
$\ldots$

- **Skip**

end