Translation Rules

High Level

$$[Expr, Param, Identifier, Field]$$

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

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

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

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

$$TransSCJProg : SCJProg \pfun (CircusProg, Framework)$$

$$\forall\, scjProg : SCJProg \mid$$
$$\quad TransSCJProg(scjProg) = (TransClasses(seqC), Framework)$$

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

$$TransClasses : \operatorname{seq} SCJClass \pfun \operatorname{seq} ClassTuple$$

$$\begin{aligned}
&\forall\, classes : \operatorname{seq} SCJClass \bullet \\
&\quad \forall\, classSCJClass \bullet \\
&\qquad TransClasses(\langle class \rangle) = TransClass(class) \land \\
&\qquad TransClasses(\langle class \rangle \frown classes) = \langle TransClass(c) \rangle \frown TransClasses(classes)
\end{aligned}$$

$$TransClass : SCJClass \pfun ClassTuple$$

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

$$TransMeths : \operatorname{seq} SCJMethod \pfun \operatorname{seq} Action$$

$$\begin{aligned}
&\forall\, meths : \operatorname{seq} SCJMethod \bullet \\
&\quad \forall\, meth : SCJMethod \bullet \\
&\quad TransMeths(\langle meth \rangle) = TransMeth(meth) \land \\
&\qquad TransMeths\langle meth \rangle \frown meths) = \langle TransMeth(meth) \frown TransMeths(meths) \rangle
\end{aligned}$$

$$TransMeth : SCJMethod \pfun Action$$

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

$$TransParams : Params \pfun CircParams$$

$$\begin{aligned}
&\forall\, params : Params \bullet \\
&\quad \forall\, param : Param \bullet \\
&\quad TransParams(\langle param \rangle) = TransParam(param) \\
&\qquad TransParams(\langle param \rangle \frown params) = \langle TransParam(param) \frown TransParams(params) \rangle
\end{aligned}$$

$$TransParam : Param \pfun CircParam$$

$$TransBlock : SCJBlock \pfun CircBlock$$

$$\begin{aligned}
&\forall\, block : SCJBlock \bullet \\
&\quad \forall\, e : Expr \bullet \\
&\quad TransBlock(\langle e \rangle) = TransExpr(e) \land \\
&\qquad TransBlock(\langle e \rangle \frown block) = \langle TransExpr(e) \frown TransBlock(block) \rangle
\end{aligned}$$

Low Level

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

- $DataMethod : MethodDeclaration \pfun$: translates data methods into an ***Oh Circus*** method
- $MethodBody : Block \pfun \seq CircExpression$: translates a Java block, for example a method body
- $Registers : Block \pfun \seq Name$: extracts the Names of the schedulables registered in a Java block
- $Returns : Block \pfun \seq Name$: extracts the Names of the variables retuned in a Java block
- $Variable : (Name, Type, InitExpression) \pfun (CircName, CircType, CircExpression)$: translates a variable
- $Parameters : (Name, Params, ReturnType, Body) \pfun \seq CircParam$: 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 }
```

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

State

$\mathit{this} : \text{ref } [\![\mathit{Identifier}]\!]_{name}\ \mathit{Class}$

state *State*

Init

State'

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

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

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

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

$\ldots$

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

$$
\begin{array}{l}
Methods \mathrel{\widehat{=}} \\
\left(\begin{array}{l}
GetSequencer \\
\Box \\
InitializeApplication \\
\Box \\
MethName(AppMeth\_1) \\
\Box \\
\ldots \\
MethName(AppMeth\_n) \\
\ldots
\end{array}\right) \; ; \; Methods
\end{array}
$$

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

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 }
```

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

State

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

$$\textbf{state}\ State$$

Init

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

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

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

$$\ldots$$

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

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

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

$$\textbf{end}$$

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 }
```

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

State

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

state *State*

Init

$State'$

$this := \mathbf{new}\ [\![Identifier]\!]_{name}\ Class()$

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

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

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

...

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

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

• $(Init\ ;\ Methods)\ \ \triangle(end\_mission\_app\ .\ IdOf(Identifier) \longrightarrow \textbf{Skip}$

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 }
```

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

State

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

state *State*

Init

$State'$

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

$handleAsyncEvent \mathrel{\widehat=}$

$$\begin{pmatrix} handleAsyncEventCall\,.\,IdOf(PName) \longrightarrow \\ [\![\ [\![\ HandleAsyncBody\ ]\!]_{Method}\ ]\!]_{MethBody}; \\ handleAsyncEventRet\,.\,IdOf(PName) \longrightarrow \\ \textbf{Skip} \end{pmatrix}$$

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

...

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

$Methods \mathrel{\widehat=}$

$$\begin{pmatrix} handleAsyncEvent \\ \Box \\ MethName(AppMeth\_1) \\ \Box \\ MethName(AppMeth\_n) \\ \ldots \end{pmatrix} ;\ Methods$$

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

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 }
```

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

State

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

state *State*

Init

$State'$

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

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

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

$\ldots$

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

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

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

end

Data Class

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

state *State*

$[\![VarName]\!]_{name} : [\![VarType]\!]_{type}$

state *State*

initial *Init*

$State'$

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

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

- **Skip**

end