Translation Rules

High Level

$$[Var, Expr, Name, Param]$$

$$SCJBlock == \operatorname{seq} Expr$$
$$Params == \operatorname{seq} Param$$
$$SCJMethSig == Name \times Params$$
$$SCJMethod == SCJMethSig \times SCJBlock$$
$$Methods == \operatorname{seq} SCJMethod$$
$$Vars == \operatorname{seq} Var$$
$$SCJClass == Name \times Params \times Vars \times Methods$$
$$SCJProg == \operatorname{seq} SCJClass$$

$$[CircAction, CircVar, CircParam, CircName, CircType, CircExpression]$$

$$CircActions == \operatorname{seq} CircAction$$
$$CircState == \operatorname{seq} CircVar$$
$$CircParams == \operatorname{seq} CircParam$$
$$CircProcess == CircName \times CircParams \times CircState \times CircActions$$
$$CircusProg == \operatorname{seq} CircProcess$$

$$TransSCJProg : SCJProg \pfun CircusProg$$

$$\begin{array}{l}
\forall\, scjProg : SCJProg \mid \\
\quad \exists\, c : SCJClass;\ seqC : \operatorname{seq} SCJClass \mid \\
\quad scjProg = c \frown seqC \bullet \\
\quad\quad \exists\, p : CircusProg \mid \\
\quad\quad\quad p = TransClass(c) \frown TransClass(seqC) \bullet \\
\quad\quad\quad TransSCJProg(scjProg) = p
\end{array}$$

$$TransClass : SCJClass \pfun CircProcess$$

$$\begin{array}{l}
\forall\, class : SCJClass \mid \\
\quad \exists\, n : Name;\ p : \operatorname{seq} Param;\ v : \operatorname{seq} Var;\ m : \operatorname{seq} Meth \mid \\
\quad class = n \times p \times v \times m \bullet \\
\quad\quad \exists\, c : CircProcess \mid \\
\quad\quad\quad c = TransName(n) \times TransParams(p) \times TransVars(v) \times TransMeths(m) \bullet \\
\quad\quad\quad\quad TransClass(class) = c
\end{array}$$

$$TransClasses : SCJClass \pfun CircusProg$$

$$\begin{array}{l}
\forall\, classes : \operatorname{seq} SCJClass \mid \\
\quad \exists\, c : SCJClass;\ seqC : \operatorname{seq} SCJClass \mid \\
\quad\quad classes = c \frown seqC \bullet \\
\quad\quad \exists\, p : CircusProg \bullet \\
\quad\quad\quad p = TransClass(c) \frown TransClasses(seqC)
\end{array}$$

$$TransMeth : SCJMethod \pfun Action$$

$$\begin{lstlisting}
\forall m : SCJMethod \mid \\
\quad \exists ms : SCJMethSig;\ b : SCJBlock \bullet \\
\quad\quad m = ms \times b \bullet \\
\quad\quad TransMeth(m) = TransMethSig(ms) \times TransBlock(b)
\end{lstlisting}$$

$$TransMethSig : MethSig \pfun ActionSig$$

$$\begin{lstlisting}
\forall ms : MethSig \mid \\
\quad \exists n : Name;\ p : \seq Param \mid \\
\quad\quad ms = n \times p \bullet \\
\quad\quad TransMethSig(ms) = TransName(n) \times TransParams(p)
\end{lstlisting}$$

$$TransParams : SCJParam \pfun \seq CircParam$$

$$\begin{lstlisting}
\forall p : \seq SCJParam \mid \\
\quad \exists param : SCJParam;\ seqParam : \seq SCJParam \mid \\
\quad\quad p = param \frown seqParam \bullet \\
\quad\quad TransParams(p) = TransParam(param) \frown TransParams(seqParams)
\end{lstlisting}$$

$$TransBlock : SCJBlock \pfun CircBlock$$

$$\begin{lstlisting}
\forall b : SCJBlock \mid \\
\quad \exists e : Expr;\ seqE : \seq Expr \mid \\
\quad b = e \frown seqE \bullet \\
\quad TransBlock(b) = TransExpr(e) \frown TransExprs(seqE)
\end{lstlisting}$$

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

State

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

state *State*

Init

$State'$

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

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

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

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

$\ldots$

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

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

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

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

$State'$

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

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

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

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

...

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

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

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

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 }
```

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

State

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

$$\textbf{state}\ State$$

Init

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

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

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

$$\ldots$$

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

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

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

$$\textbf{end}$$

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