Translation Rules

High Level

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

$$[Action, CircVar, CircParam, CircName, CircType,$$
$$\quad CircExpression, ClassDefinition,$$
$$\quad Paragraph, ChannelDefinition, ChanSetDefinition$$
$$\quad , ProcDefinition]$$
$$CircActions == \text{seq}\, Action$$
$$CircState == \text{seq}\, CircVar$$
$$CircParams == \text{seq}\, CircParam$$
$$CircParagraph ::= para\langle\langle Paragraph\rangle\rangle \mid$$
$$\quad chanDef\langle\langle ChannelDefinition\rangle\rangle \mid$$
$$\quad\quad chanSetDef\langle\langle ChanSetDefinition\rangle\rangle \mid$$
$$\quad procDef\langle\langle ProcDefinition\rangle\rangle$$
$$CircusProg == \text{seq}\, CircParagraph$$
$$Framework == \text{seq}\, CircParagraph$$

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

$$TransClass : SCJClass \pfun \text{seq}\, CircParagraph$$

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

$$TransClasses : \text{seq}\, SCJClass \pfun \text{seq}\, CircParagraph$$

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

$$TransSCJProg : SCJProg \pfun CircusProg \times CircusProg$$

$$\forall\, scjProg : SCJProg \bullet \exists\, f : Framework \bullet$$
$$\quad TransSCJProg(scjProg) = (TransClasses(scjProg), f)$$

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 *OhCircus* 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*
- *MethodsOf*(*name*) : yeilds a sequence of methods from the class *name*

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

$State'$

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

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

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

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

...

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

$$
\begin{array}{l}
Methods \mathrel{\widehat{=}} \\
\left(\begin{array}{l}
GetSequencer \\
\Box \\
InitializeApplication \\
\Box \\
MethName(AppMeth\_1) \\
\Box \\
\ldots \\
MethName(AppMeth\_n) \\
\ldots
\end{array}\right) \;;\; Methods \\
\\
\bullet\ (Init \;;\; Methods) \;\; \triangle (end\_safelet\_app \longrightarrow \mathbf{Skip}) \\
\\
\mathbf{end}
\end{array}
$$

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 }
```

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

State

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

state *State*

Init

$State'$

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

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

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

...

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

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

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

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

$State'$

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

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

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

...

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

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

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

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